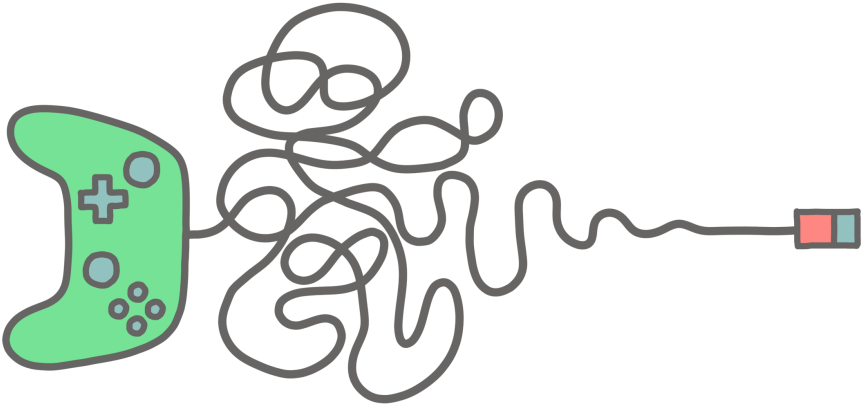
Production Point



How to
PLAN and FINISH
Your Game as a Solo Developer

Benjamin Kean Anderson

Contents

Introduction

The Potential Point Scam

After three months of hard work, my brother and I were seeing the fruits of our labors. We'd just published a prototype for our monster taming RPG (*Demonlocke*) and we were getting hundreds of emails with feedback on the game. Most of the feedback was positive and I could tell the project had a lot of potential.

We had managed the project well up to that point, but that was about to change. The potential I saw in the game had enamored me. I was on the verge of a critical mistake: one that I've seen indie game developers make time and time again, and one that would cost me years of wasted effort.

This mistake was committing to the design too soon. This early commitment caused a shift in my focus and my mentality. More specifically, a focus shift from *mechanics* to *content* and a mentality shift from *playfulness* to *industriousness*. While this shift is necessary in the lifespan of any project—more on this later—it must be done at the right time.

Because I made the shift too early, committing at this *potential point*, I started to suffer the side effects. My efforts were wasted and progress came to a sudden halt. I floundered even as I worked harder and harder. Months turned into years and I still wasn't any closer to finishing than when I'd first started. I got

discouraged, burnt out, and lost all motivation. I didn't understand what had gone wrong.

I began to wonder if there was something wrong with my approach. After all, if you work hard every day on something for years only to end up right back where you started, something must be wrong.

This wasted effort led me to watch other developers, and I soon discovered that I wasn't alone.

In his video titled "How NOT to make an indie game,"[1] the developer of *Patch Quest* describes his game development journey and his experience with the same mistake: "I started development with a whole bunch of different ideas and plans... it wasn't really clear in my mind exactly how all these pieces were going to fit together. Even so... I jumped straight in."

In his initial excitement to get started on his idea, he committed right away. This negatively affected development moving forward. Later in the video, he describes how it affected his progress: "the first year of development was disheartening. I wasn't anywhere near creating the thing that I wanted to make." Despite the rough first year, *Patch Quest's* developer continued to push through. He worked even harder and tried to rework the mechanics and better explain them. Unfortunately, the core of the game wasn't working and no amount of patching was going to fix it.

Midway through the video, he displays a rainy background and a giant block of text with the words: "I rushed the prototyping step."

He'd committed too soon, just like me, and the mistake cost him years of development time. His discouragement shows as he describes the first three years of development: "this was a really tough lesson to learn, because by now I'd spent over three years on this game, working nearly every day to develop it into something great, and yet by this point, even I didn't think it was any good."

---

1. https://youtu.be/NnI_1DOYt2A

Patch Quest was later redesigned from the ground up and the developer spent more time prototyping until he found something he could commit to. It was released on Steam in early access and has a 98% positive review score out of 460 reviews.

At the end of the video the developer remarks, "if I had spent that first year... trying out a bunch of ideas before settling on a plan forwards, then maybe I could have avoided wasting the second and third years of development."

Patch Quest is a single example cherry-picked from YouTube, but the issue is so common that Bennet Foddy, a game developer and game design professor at NYU's Games Center, was forced to redesign his curriculum in an attempt to prevent his students from making the same mistake.

In his GDC talk titled "Game a Week: Teaching Students to Prototype," Foddy says that "students tend to over commit to their seed ideas and they spend all their time building their first idea rather than trying out different ones and seeing which ideas are good."[2]

Because of this tendency, Foddy implemented a game-a-week system into his curriculum to encourage his students to create a variety of prototypes and prevent them from getting "married" to any single idea early on.

But it's not just college professors trying to warn new developers of this danger; experienced developers are trying to as well.

Tom Francis, the developer behind *Gunpoint, Heat Signature,* and *Tactical Breach Wizards,* made a five-minute GDC talk in an attempt to address the issue by encouraging new developers to "give up" on ideas. In the context of his talk, "giving up" means not committing to any single idea right away, but instead being willing to explore.

---

2. https://youtu.be/9O9Q8OVWrFA

Francis says, "developers don't generally announce the games they give up on, so from the outside it looks like they're finishing everything they start. But behind the scenes, people are giving up on things."[3]

Francis wants new developers to understand that not every game they start needs to be finished. Even experienced developers drop ideas and projects that aren't working.

Near the end of the video Francis adds to this idea by saying, "I think giving up is sometimes stigmatized as an act of self doubt, but I see it as an act of faith in yourself. When you're working on something that's showing no signs of working out, giving up on that is saying, 'I trust myself to come up with something better.'"

We need to have the faith in ourselves to "give up" on ideas that aren't working and try new ones.

After looking back on my own experience and researching other developers I was forced to acknowledge that committing too early leads to wasted money, time, and effort. This led to the realization that:

Effort ≠ Progress

Knowing exactly when to commit is paramount to making progress and finishing a game. The developers who can identify the correct point at which to commit are the ones who finish games. They don't have a special super power. They don't even work harder than you. In fact, some might work less. But when they commit to something, it's at the right moment, and their work goes exactly where it should. For these developers:

Effort = Progress

When I committed to *Demonlocke*, it was at the *potential point*. Unfortunately, this is when most developers commit to their idea, and it's a grave mistake.

---

3. https://youtu.be/Wg4SvCNQuB4

Experienced developers know to avoid the potential point. Instead, they commit to their projects at what I call the *production point*. At this point, all the major uncertainties of a project have been addressed. These developers can safely commit to the idea and know that their efforts will lead to immediate progress toward a finished game.

While there are examples of developers committing early and hard to their dream game and hitting it big, these examples are the exception and not the rule. Outliers like these suffer from survivorship bias.

If you commit too soon to your dream game, you allow survivorship bias to work against you. The chances of that single game idea being successful are slim.

However, if you explore a lot of prototypes without immediately committing, you allow survivorship bias to work in your favor. You're using it to weed out the weak ideas and hone in on the strong ones.

But why is committing at the potential point so dangerous? How can you know if your project is at the potential point or the production point? What steps can you take to make sure you commit at the correct time and what does development look like after committing? Throughout each chapter in this book, I'll address these concerns and guide you through your game's development cycle. I'll introduce the exact development cycle breakdown that I use, and give you different actions to take during each phase.

This understanding will allow your effort to result in progress. You'll become a developer who finishes games. Your effort won't be wasted, because you'll know how to leverage that effort in a way that creates progress.

I implemented this understanding while developing my next commercial game, *Tic-Tac-Tanks*. I correctly identified the production point and was able to develop and release the game from start to finish in the span of a year. There were no major roadblocks or setbacks. I completely avoided the potential point and jumped into production the moment the project was ready.

Over the course of this book, I'll teach you how to identify the production point and avoid the potential point. I'll break the game development process down into two phases and give you actions to leverage your efforts during those phases.

How to Use This Book

If you are an experienced developer with a released commercial game, this book isn't for you. You may find value in it, but it was written for first-time developers who haven't released any games yet.

I started writing this book because I saw a need. I saw myself making the same mistakes over and over during development, and I noticed other first-time developers making these mistakes as well. I wanted to create a framework to follow that would guide new developers through the gamedev process without restricting them too much. The framework would help developers avoid the big mistakes, but allow them to make and learn from the little ones.

I saw a gap in the amount of guidance aimed at solo developers and small teams. There are already books out there designed to guide studios through the development process, *Agile Game Development* and *A Playful Production Process* to name a couple, but there aren't any books for people like me. I make games primarily by myself—contracting out some tasks—and I know a lot of other indie developers who do the same. In all my reading, I've never found a book that was written for my situation. I wanted to create that book and share it with others.

In his massively popular 3-2-1 newsletter titled "Consistency, anger, and shaping the world," James Clear, author of *Atomic Habits,* says "asking what makes someone successful is like asking which ingredient makes a recipe taste

good. It's not any single ingredient. It is the combination of many ingredients in the right proportions and in the right order."[4]

Clear makes a good point. People often want a single solution to their struggles, but that isn't how the world works and it isn't how this book works either.

This book isn't an exact formula for success. If you're looking for a book by an indie gamedev guru with a secret ingredient, silver bullet, or one simple trick to becoming a successful game developer, then you're in the wrong place.

This book is a recipe.

Recipes are meant to be adjusted and iterated on. There isn't a "correct" way to follow a recipe, and having a well-made recipe doesn't make you a capable cook. For that, you need frequent and abundant practice. You need to experiment and make mistakes. These mistakes will benefit your learning process and, as long as you don't burn the house down, you'll see new ways to improve. Sometimes you won't consciously see them. Sometimes it will simply feel right to do it slightly differently. A tad less salt. A touch more milk. Does this dough have the correct consistency? Touch it and see. What does the correct consistency feel like? You'll know with practice.

Please don't treat this book like a lecture by a college professor. Treat it like a cooking lesson with grandma. As you dig in and try your own ideas, I'll guide your hand so you avoid the big mistakes like burning the house down.

I hope that this book can be a guiding recipe for you as you make games. Just like a well-made recipe, nothing in this book should be considered law. It is open to modifications as you (the chef) feel are appropriate for your situation and available ingredients. However, as a beginner, it might be wise to stick closer to the recipe at first. You can always mix things up later as you become more comfortable with the process.

---

4. https://jamesclear.com/3-2-1/november-24-2022

It may feel daunting staring into the unblinking gaze of a new learning journey. There is a desire to learn quickly, avoiding all pitfalls and plateaus. This desire is natural. Unfortunately, there is no shortcut to mastery, in cooking or gamedev, but you can learn and this book will be there for you every step of the way.

1

The Two Phases

Explore-Exploit

In 2006, two game developers and friends by the names of Kyle Gabler and Ron Carmel were working at Electronic Arts (EA) and facing a difficult decision.

They had two options:

- A: Keep their safe jobs at EA making a basic living.
- B: Quit their jobs and risk everything to start an indie game development company together.

With option *A*, Gabler and Carmel could continue to exploit an already good option, keeping their day jobs. But, with option *B*, they could explore new opportunities, earning either much *more* or much *less* than they were before.

This dilemma hides a principle that forms the foundation of this book: one that provides a solution to the problem of committing at the potential point. Mathematicians refer to this dilemma as the *explore-exploit* tradeoff.

In a later YouTube interview, Carmel would remark: "we wanted to both fail and succeed. Which we were going to do neither working at a large company."[1] Gabler and Carmel wanted the chance to make their mark on the world. They weren't interested in living an average life.

So with $10,000 in savings, they quit their jobs and started a small indie studio called 2D Boy.[2]

At first, they weren't quite sure what game they should make. After trying a few different ideas, nothing seemed to stick.

As luck would have it, Gabler had an extensive portfolio of game prototypes created during his college years at Carnegie Mellon University (CMU).

Gabler was part of a four-person team participating in the Experimental Gameplay Project. In the project, each team member would make a single game every week. The program ran for two semesters and birthed dozens of small prototypes.

One prototype towered above the rest. It was called *Tower of Goo,* and it had gained some popularity online. In the prototype, the player would construct a physics based tower by placing different joints higher and higher up the screen until the tower eventually came tumbling down. The small project was popular enough that another company was trying to clone and sell it on the mobile market. This spurred Gabler and Carmel to action, and they decided to take advantage of the success of this prototype by turning it into a full game.

From this small prototype came the massive indie hit *World of Goo.*

Gabler and Carmel may not have understood it at the time, but they completely avoided the potential point by splitting their development process into

---

1. https://youtu.be/o3sKOxZCcXo

2. https://2dboy.com

two phases: an exploration phase early in development, and an exploitation phase later.

The *exploration* phase came while Gabler was part of the Experimental Gameplay Project at CMU, and the *exploitation* phase came when they teamed up after quitting their jobs at EA and doubled down on the relative success of *Tower of Goo*.

Explore-exploit.

The other key ingredient in their story is the evaluation point, located between the exploration phase and the exploitation phase. At that point, Gabler and Carmel were able to use the information they gathered during the exploration phase to determine if they were at the potential point or at the production point.

Breaking development down into these two phases is something I've already put into practice with my most recent commercial game: *Tic-Tac-Tanks*. The two phase process made development an enjoyable experience and the whole process flowed smoothly. The contrast between my experience with *Demonlocke* and my experience with *Tic-Tac-Tanks* is what led me to the two phases in the first place.

Interestingly, I'm not the first person to break the gamedev process down into two phases. When I first proposed this two phase framework in my YouTube video titled "How to PLAN your Game as a Solo Developer,"[3] I had a few comments about another individual who uses a similar method. That individual is Mark Cerny.

Cerny's portfolio is absolutely massive. He's worked on a variety of famous game titles including *Marble Madness*, *Crash Bandicoot*, *Spyro the Dragon*, *Jak and Daxter*, *Ratchet and Clank*, *Sonic*, *Uncharted*, and *Marvel's Spider-Man*, to name a few.

---

3. https://youtu.be/NsMHicoZTzQ

In his now famous 2002 DICE talk titled "Method," Cerny also splits the game development process into two distinct and separate phases, stating that "the creation of the game is split into two phases. Pre-production, the chaotic first phase in which the game's design takes form. And production, the second phase of the project during which you build the game. These two phases are as different as night and day."[4]

As Cerny developed hit after hit, he started to notice a pattern. He realized that the first phase of development was "chaotic" as the game's design came together and that the second phase of development was simply "building" on the early design. These two phases are the exploration and exploitation phase.

It's not always easy to identify these two phases, and it can be even harder to find the point that separates them. I'll attempt to give you the tools you need to accomplish these tasks. We'll first draw clear lines between the two phases by comparing and contrasting them.

Clear Lines

Understanding the boundaries of each phase will help you keep them separate as you work.

I'm going to contrast the two phases by focusing on four main aspects. These aspects are:

1. Mentality
2. Focus
3. Feedback
4. Deliverable

---

4. https://youtu.be/QOAW9ioWAvE

Let's look at a table with each of these aspects in the context of the two phases:

	Pre-production	**Production**
Mentality	Playful	Industrious
Focus	Explore Systems and Mechanics	Create Content
Feedback	Small Group + Big Picture	Large Group + Polish
Deliverable	Vertical Slice	Finished Game

The two phases are completely different from each other. Creating a clear dividing line between them will help you focus on the phase you are currently in and prevent you from mixing them.

This table gives a nice overview, but let's take a deeper look at each aspect and how it relates to both phases.

Mentality

Your mentality dictates the way you approach each phase.

During pre-production, you should have a playful mentality. I recommend you use rapid prototyping to cultivate and maintain this mentality. Rapid prototyping involves creating paper prototypes, rough digital prototypes, and entering game jams. These prototyping methods will encourage you to explore quickly. This will be the topic for **Chapter 2**.

During production, you should have an industrious mentality. I recommend you use a routine to cultivate and maintain this mentality. Creating a balanced and consistent routine involves learning about habits and environment. A balanced routine will push you, but not so much that you burn out. A consistent routine will give you momentum and keep you motivated. This will be the topic for **Chapter 7**.

Focus

Your focus directs your attention and actions during each phase. I chose the word "focus" because there will be times when you perform actions that are outside the main focus of the phase you are in. The word "focus" allows for these exceptions.

During pre-production, you should focus on exploring systems and mechanics. I recommend you follow the fun in your prototypes to maintain this focus. Following the fun involves looking for the toy in your game, using the toy to filter new features, and using incentives to direct players to the toy. This will be the topic of **Chapter 3**.

During production, you should focus on creating content. I recommend you create a schedule to maintain this focus. You will use this schedule to add more value to the systems and mechanics you already have. The schedule will be the filter by which you evaluate new content. This will be the topic of **Chapter 8**.

Feedback

The feedback for each phase will direct your iterative efforts. This feedback will prevent you from getting stuck in an infinite exploration loop during pre-production and an infinite polish loop during production.

During pre-production you should focus on the big picture. I recommend you watch playtesters to get big-picture feedback. Get feedback from a small group of playtesters—specifically ones from your target audience. Feedback should be frequent, abundant, and accurate. This will be the topic of **Chapter 4**.

During production you should focus on polish and accessibility. I recommend you run alpha and beta tests to get feedback on polish and accessibility. You want large groups of testers with a variety of different skill levels and

on a variety of hardware devices. The goal is to widen the range of players who could enjoy your game while keeping true to its core and your vision. Feedback doesn't need to be as frequent during production but it should still be abundant and accurate. Running monthly alpha or beta tests should be enough. This will be the topic of **Chapter 9**.

Deliverable

The deliverable is the ultimate goal of each phase. The work you do during each phase should slowly move you toward this goal.

The deliverable for pre-production is a vertical slice.[5] This vertical slice will remove all the big questions about your game's design before you move to production and you will use it to evaluate your game's readiness for production. It's a crucial step in avoiding the potential point. The vertical slice will be the topic of **Chapter 5**.

The deliverable for production is a finished game. You'll release your game by following a launch plan. This plan will guide you through the launch process and give your game the best chance for success. This will be the topic of **Chapter 10.**

After launching your game, you'll support your customers through bug fixes, potential updates, and, depending on how well the game does, maybe even downloadable content (DLC). The post-launch support will be the topic of **Chapter 11**.

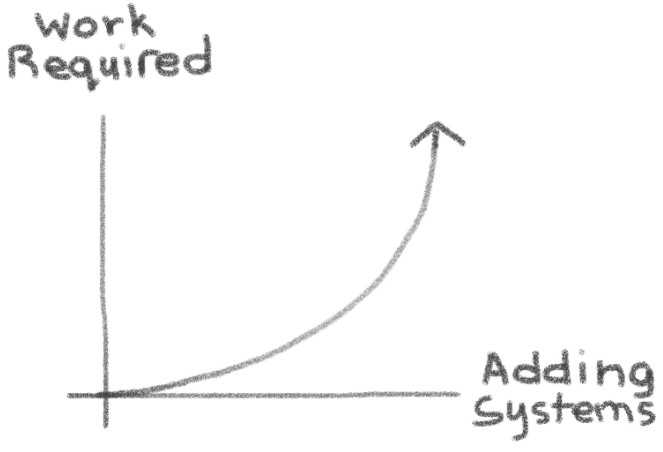
Progress During Each Phase

5. A vertical slice is like a demo of your game. We'll cover the topic in more detail later in the book.

One final thing to keep in mind as you contrast the two phases is that progress looks different depending on which phase you are in. While the production point method is designed to help you create a quality game, it's also designed to help you finish your game. Understanding how progress looks in each phase will keep you on track to that goal.

Making progress during pre-production starts off easy and gets exponentially more difficult. There are many factors that contribute to this. Program complexity is one of them, though it can be mitigated through good programming practices. Another is design complexity. As you add more systems and mechanics, it gets more and more complicated to design new mechanics within those existing systems. A small design change could have massive consequences for the overall design of the game. Regardless of the reasons, we need to keep the exponential nature of pre-production's difficulty in mind.

Here is a diagram of what progress looks like in pre-production:

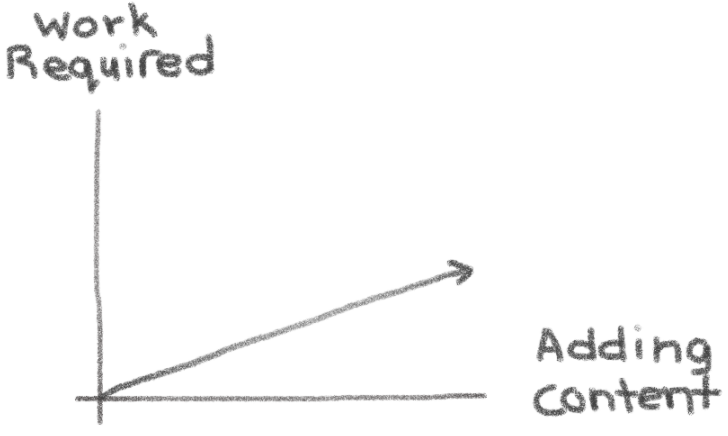


Making progress during production follows a linear curve. Your focus will be on content. Adding a second level to your game shouldn't be that much harder or take that much more time than adding the first level. You'll use the previously

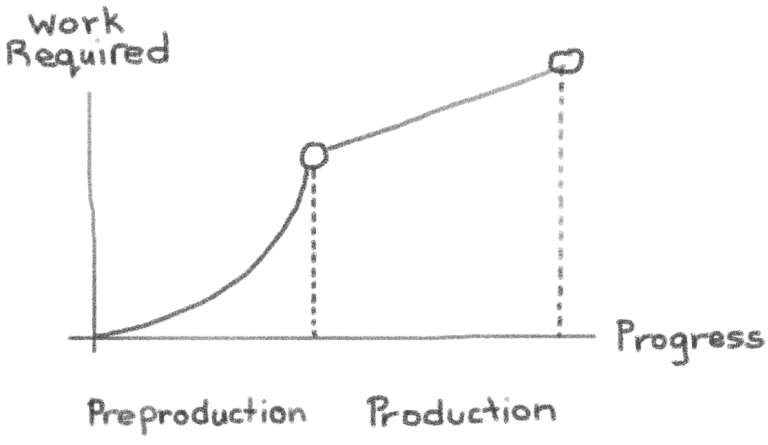
created content to make educated estimates about how long each new piece of content will take.

During pre-production, finishing your game can feel like an impossible task. Once you are ready for production, this linear curve can give you hope and help you see the light at the end of the tunnel.

Here is a graph of what progress looks like in production:



If we combine these two graphs together, we can visualize the entire development cycle:



The point (in the middle of the graph) where we switch from pre-production to production is the production point.

Exceptions to the Clear Lines

Tom Francis, the creator of *Gunpoint, Heat Signature,* and *Tactical Breach Wizards,* has a unique open-development approach. While making his games, he creates 10-30 minute "rambly" (as he puts it) devlogs and uploads them to YouTube.

In a devlog titled "Game design: the non-stick plan," Francis describes how he plans his games: "a lot of indie developers... say you shouldn't have a design document at all because it restricts your thinking and you should be open to experimentation... The more traditional way of developing games is actually have a really rigid plan because you have a huge team of people and everyone needs to be working towards the same thing... And my approach is kind of half of each. Which is, have a plan and don't stick to it."[6]

---

6. https://youtu.be/iDN7jA7hEhA

I've defined clear lines between the two phases to give you a plan—a set of rules to guide you. These dividing lines will inform your decisions and prevent you from committing too soon.

However, lines must sometimes be crossed. There are many examples of calculated rule breakers who have made a positive impact on society. These individuals teach us that there are exceptions: exceptions that we must be open to.

While reading this you may worry that sticking with the two phases will limit your ability to make systemic changes to your project during production—preventing your creative expression. What do you do when you have an idea for a new feature or mechanic and you have moved to production?

While the focus of pre-production should be the systems and mechanics, and the focus of production should be on the content, there are times when it's okay to work on content during pre-production and rework a major system after moving to production—though in the latter case, I usually recommend moving back to pre-production temporarily. How much and how often you cross these lines should correspond with your track record for finishing games. A finished game is the priority.

In Josh Waitzkin's book, *The Art of Learning*, he teaches a concept he calls "form to leave form."[7] This concept can be described as mastering the form of something until you no longer need to think about that form. Like a piano player that no longer needs to think about each note in order to play a song, the form becomes a part of who you are. Once you have mastered the form, you are free to leave it. You will know when that moment comes, when you should break the rules, get creative, and follow your gut.

This two phase process isn't perfect. Each developer is going to forge their own path to some extent. But, I truly believe that using it will be the best way for

---

7. Josh Waitzkin, The Art of Learning

beginners to start. If you are new to game development and have never finished a game, I highly recommend you stick to the process. This is the form. Once you have finished a game or two and feel comfortable with the process, I would encourage you to start breaking the rules where you feel it is appropriate. Trust your instincts. This is leaving the form.

The Production Point Method is a non-stick plan. It is a framework that will be helpful to get your game to the finish line, but it isn't the only way. The main goal of the plan is to assist you as you finish a game. As you get better at finishing games, you will learn where to break away from the plan. You will know the form and when to leave it.

Conclusion

Game development is a two phase process. Understanding these two phases in detail will help you identify them and the point that separates them. That understanding is crucial to making sure you commit at the production point and not the potential point.

Pre-production is the first phase. It's exploratory or "chaotic." During this phase focus on systems and mechanics, be playful, seek big picture feedback, and work toward a vertical slice or demo.

Production is the second phase. It's exploitative and builds on the design you established during pre-production. During this phase focus on creating content, be industrious, seek feedback on the smaller details, and work toward a completed game.

If you keep these two phases separate and commit to production at the right moment, it will make your development process a much smoother experience. You'll understand exactly what progress looks like during each phase. You won't commit to your ideas too early. And when you do commit, you'll make consistent progress towards your completed game.

A firm understanding of the differences between the two phases will also allow you to break away from them when appropriate.

Chapter Summary

- Splitting the game development process down into two phases, an exploration phase and an exploitation phase, with an evaluation point between them, allows us to avoid committing too early to an idea
- Your **mentality** during pre-production should be playful—for production it should be industrious.
- Your **focus** for pre-production should be on exploring systems and mechanics—for production it should be content creation.
- The **feedback** during pre-production should be frequent, big picture, and from small groups—for production it can be less frequent, focus on polish and accessibility, and from larger groups of testers.
- The **deliverable** for pre-production should be a vertical slice of the game—for production it should be the finished game.
- Progress during each phase looks different. Understanding what progress looks like during the two phases will help you continually make progress towards a finished game.
- There are some exceptions to the clear lines I've drawn between the two phases. As you gain experience, you will learn when to stick with the guidelines they provide and when to break away from them.

Pre-production

"The chaotic first phase in which the game's design takes form."
– Mark Cerny

2

Rapid Prototyping

"Make a little bit of everything and see what shines." - Tom Francis[1]

Quantity and Quality

In the book *Atomic Habits* by James Clear, there is an anecdote that illustrates an interesting relationship between quantity and quality. Here is the summary:

A photography professor at the University of Florida named Jerry Uelsmann decided to try something new with his curriculum. On the first day of class, he shocked his students with a crazy proposal.

He divided the class into two groups. Each of these groups would be graded based on either quantity or quality. For the quantity group, they would be graded on the amount of work they produced. For the quality group, they would be graded on the quality of their best work.

When the last day of class rolled around and the photos were graded. All of the best photos came from the quantity group. The students who had focused on producing a large quantity of work had also produced the highest quality.

---

1. Quote from a GDC talk by Francis titled "Dealing with Scope Change in Heat Signature and Gunpoint," https://youtu.be/r4-O_7wSyAQ

While the students who had focused on producing a single high quality piece had ironically missed the mark.[2]

I find this story interesting in two ways.

First, because it favors producing a lot of content in order to get better results.

Second, because I believe there is some psychology at play here.

Students worried about creating a single A-grade photo missed out on the practice and sampling period that would give them the skills to produce the quality work they sought. That single photo was their one shot at getting an A. They didn't want to screw it up.

Students who didn't care about creating a perfect photo were able to practice and explore during that sampling period. They didn't worry about any particular photo being perfect. They had plenty of photos on which to experiment and learn. The only way to screw up would be to not take photos.

It's easy to let other people, circumstances, or even your own perfectionism place you into the group that focuses on quality.

Which group are you in right now? Which group do you want to be in? Consider these questions as you start pre-production.

Sokpop (Sock Puppet in Dutch), a small four-person studio from the Netherlands, took these questions seriously when forming their game studio.

Back in 2017, they committed to the quantity group in a rather unique, and slightly crazy, way. They challenged themselves to make and release a new game every single month.[3] No, not merely a game jam project, but a finished game.

Since the inception of this challenge, they've released *over a hundred* games on Steam. They've also created a Patreon account where fans can support their

---

2. James Clear, Atomic Habits.

3. At first, they were making two games a month, but later reduced it to one a month.

work and gain access to the monthly game releases. At the time of writing this, their Patreon generates more than $8,500 USD a month.

This is an extreme example of prioritizing quantity, but Sokpop also produces quality titles. Over the years, they've improved their craft and released a few indie hits. My favorite is their most recent hit, *Stacklands,* a game about stacking cards to build a village and community. The game currently has over 14,000 reviews on Steam (with a 97% overwhelmingly positive score).

In a Sokpop devlog titled "How my indie game blew up (Stacklands devlog),"[4] the developer talks about his mindset while making *Stacklands:* "Having a mindset of trying to make a good game is really overwhelming. You're just going to second guess every decision you make. It's way better to start working on something and decide if it is good or bad after you've made it."

The stress of trying to create a single perfect A-grade photo or "dream game" can interfere with your ability to create something in the first place.

A focus on quantity will get you in the right mindset. It's much easier—and wiser—to create an abundance of games and pick out the noteworthy ones later.

I'm not going to recommend you follow in Sokpop's footsteps exactly. They've managed to make their methodology work, though it's not something I would attempt myself. However, it does serve as an example for prioritizing quantity, and they aren't the only ones doing it.

In the **Introduction** chapter I quoted Bennet Foddy, a game designer and professor at NYU's Games Center. Foddy was struggling with students over committing to their first game ideas. He saw this as a major problem that needed to be addressed.

Foddy came up with a rather radical solution. He introduced a new curriculum that would require students to create a game each week and present it to

---

4. https://youtu.be/gqVaIDANWgU

the rest of the class. The game didn't have to be bug free or even good, but it did have to attempt to answer some previously unanswered question about design.

This curriculum encouraged students to explore and prevented them from getting locked into any single idea early on. Each student was graded based on the number of weekly submissions, but not the quality of those submissions. Unlike the photography professor at the beginning of this chapter, Foddy put every student into the quantity group.

This "game a week" idea originally came from a group of four students at Carnegie Mellon University (CMU). Back then, it was called the Experimental Gameplay Project. Remember Kyle Gabler and *World of Goo* from **Chapter 1**? He was one of the four students.

The "game a week" idea went on to inspire Rami Ismail of Vlambeer to write an article about it on Gamedeveloper.com. Ismail started to see the value in producing a high number of games instead of focusing on a single great game. In his article he wrote: "that's when I realized that design experience isn't the size of your games, or even the scope...it's in the number of project's you've been through."[5]

Ismail's article inspired Ojiro Fumoto, the creator of *Downwell*, to try the "game a week" challenge as well. Fumoto did it for thirteen weeks. By week thirteen he'd found a prototype that felt fun to play, so Fumoto decided to work on it for more than a week. From it, *Downwell* was born and later published by Devolver Digital. It now has over 6,000 overwhelmingly positive reviews on Steam.

Small indie studios aren't the only ones focusing on quantity during pre-production, Nintendo also has an established culture of rapid prototyping. In a GDC talk titled, "ARMS: Building Mario Kart 8 Insights into a Showcase

5. https://www.gamedeveloper.com/audio/game-a-week-getting-experienced-at-failure

Nintendo Switch Fighter," Nintendo developer Kosuke Yabuki states that "at Nintendo we're always prototyping new games... very few of these prototypes will ever see the light of day."[6]

Keep in mind that this rule—producing a large number of games without being worried about quality—is only for pre-production. Quality does matter a lot, but just not yet. Later, I'll show you how to take a prototype with a solid foundation and polish it until it shines. For now, focus on exploration through quantity.

It's one thing to know that you should be producing a large amount of prototypes, but knowing it doesn't make it easy. You have to understand the purpose of creating quantity, and the tools you have at your disposal, in order to make it happen.

The purpose is to explore different design ideas. Sure, you could take a single prototype and copy-paste it until you had a hundred copies. That would be "quantity" but it defeats the purpose. You haven't explored a variety of ideas. You've merely duplicated the same idea. Each new prototype should answer a unique design question.

It's not easy to make a lot of prototypes, but there are actions you can take to make it possible. Here are the actions I recommend:

- Paper Prototyping
- Digital Prototyping
- Participating in Game Jams

Paper prototyping is the fastest way to get your design ideas into a tangible and, more importantly, testable form. Paper prototyping in the context of this book won't merely involve paper, but can also include toys or crafting materials.

---

6. https://youtu.be/cPdbTtR0nOo

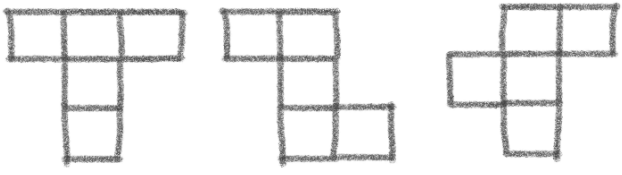
Digital prototyping is where you go when you start to reach the limitations of paper prototyping. You should still have a non committal approach while making digital prototypes. Use filler graphics and throwaway code. Stay loose and playful like a painter slashing large abstract shapes on a canvas to find a strong composition before committing to small details.

Participating in game jams is a more extreme form of digital prototyping. It gives you more pressure but also provides a supportive community. The community will usually give feedback on your ideas, giving you more information with which to evaluate their feasibility and marketability.

These three actions will opt you into the quantity group and help you maintain that playful mentality that is crucial for pre-production.

Paper Prototyping

A pentomino puzzle is a small puzzle game played with twelve blocks. These blocks form various shapes using five equally sized squares connected at the edges.



Looking at these shapes might remind you of something, and for good reason.

Alexey Pajitnov, a computer engineer and video game designer from Moscow, had his own pentomino puzzle. As he was playing with it, he noticed the way the pieces fit together and thought it was oddly satisfying. The pieces and puzzle inspired him to create his own version of the game, but on a computer.

He simplified the game's code by reducing the amount of blocks from twelve down to seven and by using tetrominoes—blocks made from four squares instead of five. He named his small prototype *Tetris* because *tetra* means "four" in Greek and his favorite sport was tennis.

It's impossible to measure the impact of Pajitnov's game on the world. The Game Boy version of *Tetris* alone sold thirty-five million units and the game holds the Guinness world record for number of official platform releases.

This world-wide phenomenon all started with a Russian adult playing with a small toy puzzle.

I have four young kids at the time of writing this book. My oldest child recently turned eight and my youngest is a few months over a year. As a father, I enjoy watching my kids play, and I've learned a lot about it from them.

The other day I was watching my six-year-old play with little paper cutouts. She was constructing a narrative around her imagined characters.

Her play often explores deeper topics like love, betrayal, family, and death. In one narrative, she described a mother who died in childbirth, leaving the child as an orphan—orphans are currently her favorite type of character. These topics are heavy, but play gives her a safe environment to explore them in. As she explores them, she learns more about how she might handle a situation like that. She learns about the world and explores her place in it.

This type of play is a skill we are all born with. Unfortunately, It's also one that we often unlearn as adults. Play is a fundamental skill of game design. If you want to design games, you have to use and develop this skill. You must allow your inner child to come out. Get out the crayons, scissors, glue sticks, and paper. Create a crafting space that allows you to explore your ideas. If you need to, get a child to come help you. They are experts.

As you explore your ideas in a physical form and talk through each step of the idea, you will learn about it and remove or discover uncertainties. We've established that designers are often over confident in their ideas early on and

commit hard too soon. Playing through ideas in this way can remove that arrogance and ignorance. It will point you to the flaws. Once you have found them, you can start addressing them.

Expressing these ideas on paper or with toys allows you to explore them quickly and with low commitment.

I have a desk in my office piled high with cardboard, tape, scissors, string, dice, paper, and lots of 3x5" note cards. I spend time at this desk every day, either creating new prototypes, or adjusting and playtesting previous ones.

I've been able to explore hundreds of ideas already without writing a single line of code.

If you aren't paper prototyping yourself, commit to starting. Jumping right into digital prototypes will cost you time and money. As indie developers, we need to make the most of those resources. We don't have a lot of them.

While paper prototyping is quick and easy to jump into. There are some drawbacks. It requires a lot of cognitive work. You'll have to do all the calculations for damage or hit chances. This is something computers do well. It's also hard to know how the game will feel to control in a digital environment. While paper prototypes are almost always an excellent place to start, it's important to move on to digital prototypes as soon as you start to reach the limits of paper prototypes.

Digital Prototyping

As a teenager, I made hundreds of small prototypes in GameMaker. The software has come a long way since I first learned it, but it's still amazing for making quick digital prototypes. Most of my prototypes were attempts to clone my favorite games using my extremely limited knowledge of programming.

I was inspired by *Old School Runescape* and *Morrowind* to make a small platformer called *Deep Magic* that implemented role-playing elements like a

leveling system and spells that could only be cast once you reached a high enough level.

I was interested in *StarCraft,* so I made a fast-paced real-time strategy game (RTS) called *Ancient War* with tiny pixel art characters and terrible pathfinding.

These were the two main games I finished.

But, surrounding these two completed games were hundreds of unfinished prototypes. I look back on that part of my life with fond memories. I spent a lot of time exploring, experimenting, and having fun. I think I learned a lot and didn't even realize it.

Now my game engine of choice is Godot engine, but for some prototypes I still find myself using GameMaker or GDevelop because they don't require code and allow me to get a digital prototype up and running in a matter of minutes.

Using the right tool while making digital prototypes is important if you want to be fast. Smaller engines like GameMaker, Construct, Pico-8, or GDevelop are great for rapid digital prototyping. The indie hit *Celeste* originally started out as a prototype made in Pico-8. While learning one of these engines isn't required—Godot and Unity can be used for rapid prototyping—you might consider it to expand the tools available to you as you prototype.

There is a feedback loop that exists between an idea and the realization of that idea sitting in front of a player. The shorter you can make the duration between these two events, the better.

In a GDC talk on YouTube titled "Dealing with Scope Change in *Heat Signature* and *Gunpoint*," Tom Francis asks this question: "how much do I have to make before I know if this works?"[7]

---

7. https://youtu.be/r4-O_7wSyAQ

Francis is asking what the minimum amount of work is that he can get away with as a game developer before knowing if an idea is a good one. Ask yourself this question with any new ideas. The answer to this question will help you shorten the feedback loop between that idea in your head and a prototype in front of a playtester.

Don't get caught up in creating clean code or beautiful art when making these prototypes. If you find yourself cleaning up your code or spending a few hours on the art, you are likely over committing to your idea. You've slipped into production at the potential point. What exactly do you need to make in order to test your idea so you can move on to the next one?

Sometimes it can be hard to finish and test your early prototypes. Luckily, there is a solution to that problem.

Participating in Game Jams

In August 2013, three friends decided to team up and make a game for the 27th Ludum Dare game jam competition. The team consisted of Ari Gibson (an animator), William Pellen (a web designer and hobbyist game developer), and Rohan Fraser (an animator as well).

The theme for the game jam was "10 seconds." The three creators decided to make a game where the protagonist (an insect knight) had to eat every ten seconds or starve to death. They managed to finish their game just in time to submit it for the jam.

It was a simple top-down perspective game with a striking art style. The main protagonist would walk around the small world killing bugs. When a bug died, it would drop a cherry. These cherries would sustain the protagonist for another ten seconds.

The game wasn't that well designed, though. They published it on the Newgrounds website and, at the time, it received an overall rating of one out of five stars.

Despite the poor reception, this small jam game would go on to inspire one of the most influential and critically acclaimed indie games of our time.

About a month later, the small team decided to participate in another game jam. This time they chose the 2013 Indie Speed Run game jam. The theme was "wealth," and the game they created was called *Tomb Cat.* It was a fast paced platformer with tight controls. You played as a mother cat collecting money to support her kittens. They weren't able to finish the game in time for the jam, but managed to finish it later and uploaded it to Newgrounds. This game was better received than their first one, getting four out of five stars.

With each new game jam, the team (Team Cherry, as they would call their studio) was refining their skills.

Soon after the Indie Speed Run game jam, they entered the April 2014 Ludum Dare game jam. The theme for this jam was "Beneath the Surface." Something about this theme inspired Gibson, the animator, to return to their first game and its protagonist. In a Game Informer article titled "The Making of Hollow Knight," Gibson said, "we thought of the little insect knight exploring a deep, old kingdom beneath the surface of the world, and everything kind of snowballed from there."[8]

Team Cherry once again failed to meet the game jam deadline, but the strong sense of atmosphere and world inspired by the "beneath the surface" theme kept them working even past the jam deadline. They would later finish and release the game as *Hollow Knight*.

---

8. https://www.gameinformer.com/2018/10/16/the-making-of-hollow-knight

I find the early progression and development of *Hollow Knight* to be fascinating. The game has sold millions of copies, it has over 200,000 reviews on Steam with a 97% positive rating, and the journey started with a one out of five star jam game on Newgrounds.

Despite the slow start, Team Cherry continued participating in game jams and you can see the progression towards their dream game with each jam. With their first jam game, *Hungry Knight*, they established the art style, atmosphere, and protagonist. With their second jam game, *Tomb Cat*, they refined their design skills and established the tight platforming controls and mechanics. With their third game jam they were Inspired by the "Beneath the Surface" theme to create their world.

With each game jam they developed their skill sets and workflow. Each new game was a completely new genre and experience. They didn't hyper focus on the same ideas. They tried new strategies and found what worked for them, cherry-picking (to make a terrible pun) the ideas that worked.

After accumulating a list of ideas that worked, they compiled it all together into one of the best indie games ever made.

While they didn't always finish their games in time for the jam, they did always finish and they always posted them publicly to get feedback—even when that feedback was negative. The deadline and community created a system that encouraged them to explore, but also to complete and evaluate.

Participating in game jams is a common strategy for success during pre-production. Game jams are game making factories that pump a variety of unique ideas. They have multiple systems in play to foster creativity and collaboration. They can also be an easy way to get your ideas in front of other players.

The first system at play is called timeboxing. Jams have a specific time constraint. Most range between forty-eight hours and one month. My preferred length is one week. Before I had children, I liked the forty-eight hour constraint, but with four kids that isn't reasonable for me anymore.

Be careful, though. Timeboxing can have negative effects. Some people will push themselves too far and overwork themselves because of the time constraint. Alternatively, some people manage healthy work habits and scope their ideas to fit into the time constraint. Make sure that you are following the example of the latter group. Get plenty of rest. Go on walks. Maintain a healthy diet. Don't overwork yourself for a game jam. It's better to foster a playful mentality. Make sure you are having fun.

Another system at play is the theme. This constraint forces developers outside their comfort zone. You can't always make the same types of prototypes in your favorite genres. You need to come up with an idea that fits well into the theme. Forcing you outside your comfort zone will encourage creativity and variety. It functions as a reminder for the purpose of rapid prototyping—to explore a lot of ideas and maintain a playful mentality.

The community is another system in the equation. Game jams encourage collaboration and teamwork. When you have a limited amount of time, working with other creators can help increase your games content and polish. Working with a lot of different people will create an environment where you can develop teamwork skills and find the people you enjoy working with the most. Forging these relationships can be valuable later. Who knows? Maybe your small team will become the next Team Cherry.

Some jams also have a bit of competition. Too much competition can be bad, but a little bit can be a useful tool to motivate.

All of these systems combine into an amazing game creation experience. Many big indie hits have come from game jam games—*Hollow Knight* being one of the most famous. Both of my commercial releases, *Endhall* and *Tic-Tac-Tanks,* originally started out as jam games. If you want to make, finish, and release games commercially, you should be participating in game jams.

If you are having trouble finding jams to participate in, start with Itch.io. You could also check out the Global Game Jam, the Train Jam, and the Ludum Dare. Your local community may also have frequent game jams.

Because I use the Godot game engine, I love participating in the Godot Wild Jam. It happens every month and lasts for ten days. You can find it on Itch.io as well.

I recommend doing at least one game jam a month during pre-production. If you have a game in production, you might consider taking breaks from your main project every once in a while to participate in game jams. Production can be a lot of work, and short breaks to work on jam games just might be the refueling your creative engine needs.

You could also attempt the "game a week" framework discussed at the beginning of this chapter. "game a week" is a self-imposed weekly game jam. I wouldn't recommend it for everyone, though. Know your own limits and don't push yourself past them.

Conclusion

Keeping a playful mentality during pre-production is hard. It's surprisingly easy to slip into a more perfectionist mindset. Designers want so badly to put themselves into the photography professor's quality group—focusing on a single perfect photo or dream game. However, that course of action won't help you explore and find the very quality you are looking for.

To explore well, you need to test a lot of ideas quickly. This is done by paper prototyping, digital prototyping, and participating in game jams. These three actions will allow you to prototype rapidly, forcing you into the quantity group and nudging you towards the playful mentality required for pre-production.

Chapter Summary

- Focus on quantity over quality during pre-production.
- Use paper prototyping to rapidly test your ideas.
- Move to digital prototyping when you reach the limits of paper prototyping.
- Use game jams to explore a variety of ideas and get them in front of players.

3

Follow the Fun

From Prototypes to Fun

In 2013, ten game developers at Nintendo got together with the goal of creating a new game that didn't fit easily into existing genres. Even though Nintendo already has established intellectual properties (IPs) like *Mario, Zelda,* and *Pokemon,* the company is always looking to create new IPs and innovate on their existing ones.

This small ten-developer team joined forces to create over seventy design proposals and a number of different technical prototypes.

One prototype in particular stood out above the rest. It used cubes for the characters so they jokingly referred to it as the "Tofu Prototype." It was multiplayer and had two teams. Each team had a color, either black or white, and the tofu cubes could shoot black or white spheres out to paint the battlefield and obtain territory for a set time limit. The team with the most territory at the end of the time limit would win. Players could also shoot paint at opposing team members to defeat them.

The most interesting aspect of the prototype, however, was the hiding mechanic. You see, as the players painted the battlefield, their own character would blend into the color of their painted territory, obscuring them from other

players on the map. If they painted a new area, though, it could reveal their location to other players. This mechanic was instantly fun and despite the terrible graphics of the prototype, the team knew they had something special on their hands.

The "Tofu Prototype" would later become *Splatoon,* one of Nintendo's first wildly successful new IPs in years.

Hisashi Nogami, the producer on *Splatoon 2,* said, "while our end goal was to create a new game, the first target we aimed for was creating a new type of play experience."[1]

This focus on the play experience is built into the culture at Nintendo. Nintendo's former president, Satoru Iwata famously said, "video games are meant to be just one thing. Fun. Fun for everyone."[2]

While not all games are designed to be fun but instead focus on narrative or some other core, the principle is the same. Follow the core experience of your game. If your game's core is something other than fun, replace the word "fun" throughout this chapter with your intended experience.

Most game developers know that they need to focus on the core, but in the heat of development, it can be easy to lose sight of what that is.

I learned this lesson the hard way while working on *Demonlocke*. Early on in development I started creating content—detailed pixel art backgrounds and creatures. It was exciting and stimulating at first, but later I ended up discouraged when I realized the core of my game wasn't engaging and I would have to start over—potentially making all that content obsolete.

---

1. https://youtu.be/2VwTVIEge1g

2. https://www.gdcvault.com/play/1019910/Disrupting

If you don't follow the fun right from the beginning, and instead focus on aesthetics or beautiful code, you risk wasting time, effort, and money on a game that may never deliver on your intended experience.

Following the fun will guide your focus to systems and mechanics during pre-production—not creating content. You'll get to the fun before you move on to production and investing time, effort, and maybe even money into your game.

But what actions can you take to make sure you are following the fun?

Here are the actions I recommend:

- Look for the toy.
- Use the toy as a filter for new features.
- Use incentives to direct players to the toy.

The toy is the element of the game that is engaging to play with no matter what rules, goals, or graphics surround it. A ball is an excellent example of a toy.

Toys also have engaging actions: verbs. These actions are the most interesting thing to do with the toy. Using our ball example, some engaging actions would be throwing and catching it. Highly engaging toys have a multitude of engaging actions. These actions should be used to direct your design and filter any new features.

As you isolate the toy and engaging actions, you can start to add incentives to encourage the player to perform and master the actions that create the experience you want. These incentives include rules and goals. Again with the ball example, a rule might be "don't carry the ball while moving, bounce it," while a goal could be "get the ball into the hoop to score points and win the game."

Following the fun can be hard, but these actions will keep you on the right track. As you follow the fun, you'll maintain a focus on the systems and mechanics of the game. This is the focus of pre-production.

Look for the Toy

In his book, *A Playful Production Process: For Game Designers (and Everyone)*, Richard Lemarchand gives this advice to new designers: "begin your digital prototyping by building toys—small, simple, playful systems."[3]

An ideal toy is easy to play with, but contains depth. Let's return to our ball example. Balls are merely rubber spheres filled with air, and yet, how many games and sports use them? There are an infinite amount of actions you can do with a ball, and a large number of those actions are engaging. The toy of your game should be similar. You want lots of potential actions, many of which are engaging—though you probably won't use all of the actions, you'll focus on a few.

While writing this book, I joined a Godot game jam and made a deck building game in just six days.

At first, I struggled to find my toy. I wasted most of my first day programming what I thought would be the interesting part of the game—a system where units moved through a lane to attack an enemy on the other side of the screen—but it turned out to be a dud.

On the second day I was able to explore ideas on paper and I got closer to the idea that would ultimately be the final game—base defending—but I still hadn't found my toy.

---

3. Richard Lemarchand, A Playful Production Process: For Game Designers (and Everyone)

It wasn't until the third day, when I started prototyping digitally, that I found the toy of my game. I made some small creatures that were controlled by physics and created a simple fireblast spell to cast on them from above. Casting the spell would send the small creatures flying across the screen. It was fun, even with terrible graphics and without rules or goals. I had found my toy.

Development went smoothly from that point on. I still had some obstacles to face, mainly dealing with how to scale the game, but the core of the game was already fun so I knew I had a strong foundation. This set a fire in me that gave me the drive to work through any difficulties that came up.

It's not always easy to identify the toy, though. I got a bit lucky in this example.

The digital prototype from the third day didn't have any incentives, like rules and goals. I believe that the absence of incentives made it easier to find my toy. Rules and goals direct players toward specific parts of the game. When looking for a toy, those rules and goals may point players in the wrong direction. Players may be encouraged to engage with unfun parts of the game. However, in the absence of rules and goals, players can explore until they find the action that is the most fun.

If you are looking for the toy in an existing game, remove the rules and goals and try to figure out what the most fun thing to do is. This will help you identify the toy.

Once you've identified your toy, you can use it to guide the rest of development.

Use the Toy as a Filter for New Features

Recently, I've been working on a grid-based tactics prototype, and I noticed that the engaging actions for the toy fell under two main categories:

1. Manipulating the positional state of the board.

2. Dealing massive damage to multiple enemies.

Comboing these two actions was interesting and exciting. The prototype shined brightest at these combo moments.

These actions make sense in the context of this prototype. The game is like chess in that you have units on a grid, but unlike chess you only have a single unit and you are fighting multiple enemy units. The imbalance of one versus many creates the interesting part of the game: using strategy to reposition and hit a lot of targets for massive damage.

This information gave my design a clear direction. After identifying this, I immediately adjusted the prototype to reflect this direction. Any skill or spell that didn't fall under one of these action categories was immediately cut from the game and any new feature I wanted to add to the game was filtered through the lens of these actions. The prototype became immediately more fun both for me and for my playtesters.

Make sure that you can explicitly name the engaging actions of your toy. If you can't describe them to someone else, you don't understand them well enough yet.

Once you fully understand your toy and its engaging actions, incentives become much more important. A well-placed incentive will encourage the player to perform the most interesting actions. Games that do this well last for years, decades, or even centuries.

Use Incentives to Direct Players to the Toy

In the late 1800s, a young Canadian P.E. instructor at Springfield College by the name of Dr. James Naismith had a problem. During the warmer months of the year, his students had plenty of engaging physical sports to choose from, all of which could be played outside. Unfortunately, during the colder months

of the year, Naismith struggled to find engaging sports his students could play indoors.

Naismith was tasked with creating a game that could be played indoors and wouldn't have as much contact as other sports. He spent his time coming up with several invented games to try out, much to the chagrin of some of his students.

One day, he asked his students to play a new game he had created, taking inspiration from *Duck on the Rock*—a game from his childhood about tossing rocks.[4] The students gave the simple game a try and enjoyed it. Nobody knew it at the time, but the game would go on to become an Olympic sport and a multi-billion dollar industry enjoyed by millions of players and fans around the world. The students wanted to call it Naismith's Game, but he reportedly declined, saying "we have a ball and some baskets, why not call it Basketball."[5]

As a young boy, basketball was a big part of my life. I remember watching the Utah Jazz and the Chicago Bulls in the playoffs at my grandmother's house. I would play for hours with my cousins—though most of them were better than me. My dad welded together a hoop and put it on an old cement pad behind my house. During the winter I would shovel thick blankets of snow in order to play. The ball would get wet and cold as I dribbled it. My hands felt like ice. I kept playing, though. Basketball was my first introduction to games.

I still have a love for basketball and enjoyed studying its history while writing this chapter. One part in particular stood out to me.

When Naismith originally proposed the game to his students, it had thirteen rules. Out of these thirteen rules, the third rule stands out to me:

---

4. https://naismithbasketballfoundation.com/james-naismith-life/

5. https://basketballplayers.omeka.net/exhibits/show/frank-mahan/introduction-to-the-game

"Rule 3: A player cannot run with the ball, the player must throw it from the spot on which he catches it, allowance to be made for a man who catches the ball when running at good speed." - Naismith Basketball Foundation[6]

This rule encourages players to throw the ball—a fun action—and directs them away from carrying the ball—an action that isn't as much fun.

Many sports have rules that prevent the player from carrying the ball. Even in examples where you can carry the ball, like American football, the ball is shaped in a way that makes it easier to throw and more satisfying to carry. These rules incentivize the players to engage with the toy in the most interesting way.

As a design exercise, consider creating your own list of actions that are engaging to perform with a ball. Consider also creating an additional list that contains actions that aren't engaging to perform with one.

Making similar lists for your own games will give you ideas for rules to keep and rules to avoid. Pick the rules that drive players towards the most fun parts of the game and away from the boring parts.

Rules are the first element of incentives, but games also incentivize their players by giving them goals. Here is an example from the same thirteen original rules for Basketball:

"Rule 8: A [score] shall be made when the ball is thrown or batted from the grounds into the basket..." - Naismith Basketball Foundation

Naismith made a simple goal to guide players toward the fun part of the game. Scoring involves getting the ball through the basket and the team to score the most points during the duration of the game is the victor.

As you combine rules and goals you create strong incentives that encourage your players to engage with your game in fun and interesting ways and avoid the boring ones.

---

6. https://naismithbasketballfoundation.com/about-basketball/naismiths-rules-for-basketball/

Conclusion

Finding the toy, using it as a filter for new features, and directing players toward it with incentives will point you toward the core experience of your game. You'll be able to focus on the systems and mechanics during pre-production and defer the creation of large amounts of content until production.

Unfortunately, because games are complex, sometimes there will be unintended consequences as you add features and incentives around your toy. In the next chapter I'll show you how to use playtesters to identify problems and iterate on your design to address them.

Chapter Summary

- Start your design by following the fun.
- If you don't find the core of your game from the start, you'll risk wasting time, effort, and money.
- Identify your toy and its engaging actions.
- Use the toy and engaging actions to direct your design.
- Use incentives to guide players to the most interesting parts of the game

4

Watch Playtesters

Playing a Silent Piano

I began taking piano lessons from my mom at about eight years old. I took enough lessons to be able to play well, but didn't practice enough to ever be great.

My oldest daughter started taking lessons a few years back, and I've been able to use my knowledge to help her practice. We have a cheap digital piano that has pressure sensitive keys and a volume dial.

The other day I challenged myself to learn one of my daughter's songs, but with the volume all the way down. The song in question was rather trivial for my level, but because the volume was off, I struggled to play it. I didn't have access to the immediate audio feedback indicating when I hit a wrong note. I managed to stumble through the notes, but it was unnecessarily difficult. I kept glancing at the keys in an attempt to gain more information about how I was doing. Even glancing down, I couldn't be sure I was playing the correct notes. The whole experience gave me a strange sense of unease.

Having the volume off was unsatisfying and made me uncertain, but it could have been worse. What if I had been blindfolded as well? Learning the song would have been impossible.

According to *The Myth of Experience*, a book by Robin Hogarth and Emry Soyer, my offhand piano challenge would be classified as a *wicked* learning environment.[1]

In a wicked learning environment, feedback is delayed, biased, scarce, and inaccurate. Causes are not directly tied to their effects. The systems and rules change constantly, and stability is short-lived. Wicked learning environments are not conducive to learning, and "experience" gained in these environments is unreliable.

If playing the piano with no sound is a wicked learning environment, what about when I play with the sound on? What type of environment is that? Hogarth and Soyer propose that this type of learning environment is called a *kind* learning environment.

In a kind learning environment, feedback is immediate, unbiased, abundant, and accurate. Causes are directly tied to their effects. The rules rarely change and the systems surrounding those rules are stable. Kind learning environments are conducive to learning and experience is both valuable and reliable.

These two environments exist on a continuum. No learning environment is completely kind or completely wicked.

In my piano story, I could have easily turned the volume back on in order to go from a wicked learning environment to a kind one.

The feeling of unease I got while playing the silent piano wasn't new to me. I've felt that same unease in the past while making games. It can be hard to know if you are playing the right "notes" as you make your game.

Here is a specific example of feedback I got on *Demonlocke* a few years back: "add a victory song and a victory screen."

Suggestions like this come from a kind heart, but a single suggestion in isolation can be hard for the designer to interpret. The playtester is suggesting a fix,

---

1. Robin Hogarth and Emry Soyer, The Myth of Experience

but what exactly is the problem to begin with? Do victories feel anticlimactic to this player? Are they confused about what happened at the end of the battle? Do they feel that their hard fought victory was not rewarded?

When you get a suggestion like this, you are forced to work backwards and derive the original experience.

Game development has both kind and wicked elements to its feedback environment. Luckily, just like my piano analogy, you have some control and can take steps to move the feedback toward the kind end of the continuum. You can turn the volume back on.

To do this, I recommend three basic actions:

1. Work closely with playtesters.

2. Get reliable feedback through video.

3. Work with a playtesting partner.

To work closely with playtesters, you'll create a designer-player relationship with them. To get reliable feedback through video, you'll ask playtesters to record themselves and then watch and listen to the way they react as they are playing. To work with a playtesting partner, you'll contact a trusted family member or friend and ask them to try your prototypes.

Let's cover each of these actions in more detail.

Work Closely With Playtesters

In the early spring of 2015, Anthony Giovannetti and Casey Yano joined forces to create a small game studio named MegaCrit Games.

The two set off to build a card game that would combine the popular roguelike genre and the growingly popular deck building genre.

Their first prototypes looked rough, but were already fun to play. Unfortunately, they had a big problem.

The playtesters didn't know when to use their attack cards and when to use their block cards. On the player's turn, they could play either an attack card or a block card. If the enemy was defending or using a buff, it made sense to attack. If the enemy was attacking, it made sense to block. However, the enemy's attacks were random, preventing any kind of battle strategy.

To solve this, the team worked closely with their playtesters until they fully understood the problem. Giovannetti describes this designer-playtester relationship in his GDC talk titled "*Slay the Spire:* Metrics Driven Design and Balance," by saying, "we decided early on that we wanted to have close engagement with our fanbase and try to make the barrier between us and them as low as possible."[2]

As they watched and communicated with their playtesters, they started to approach a solution to their problem.

Their first attempt was a "next turn" feature. This feature allowed players to hover over an enemy and see what attack it would be using on the next turn—letting players plan ahead and play cards to counter enemy strategies. It was a promising start.

It had a glaring flaw, though. The team discovered this flaw while showing the game over video calls. In livestreams, viewers would have to ask streamers to hover over each enemy every single turn in order to see what the incoming attacks would be. The studio knew that streamers would be an important part of their game's potential growth (spoiler alert: they were right) so they had to find another solution.

---

2. https://youtu.be/7rqfbvnO_H0

This led them to the game's current "intent" system. In this system, each enemy would have a small icon above its head representing its intent. This intent could be to attack, to defend, to buff itself, or some other action.

The new system gave playtesters just enough information to form strategies, and since the intent of the enemy was displayed openly above its head, it also provided that same information to any viewers watching through a stream.

With the new intent system in the game, development proceeded quickly. They released their game, *Slay the Spire*, on Steam in early access form in November of 2017.

Even after launching in early access, the small team continued watching and listening to their players. Giovannetti says, "I would often watch various streamers on an additional monitor while I was working. This let me directly listen to what the popular voices in the community felt about various cards or enemies...streamers are always talking to their audience, so you're getting their internal thoughts constantly."[3]

Slay the Spire had a slow start, but would go on to sell 1.5 million copies by March of 2019—under two years after its launch in early access.

This low barrier between the vision of the developer and the experience of the player moved *Slay the Spire*'s feedback environment from the wicked end of the continuum toward the kind end. It turned the piano volume back on. The team at MegaCrit were able to develop a close relationship with their player base, allowing two way communication and a better understanding of the problems players were experiencing.

It's clear that video feedback was a part of this developer-player relationship. Let's explore this topic in more detail.

---

3. https://youtu.be/7rqfbvnO_H0

Get Reliable Feedback Through Video

"The mouth may lie, alright, but the face it makes nonetheless tells the truth." - Friedrich Nietzsche

Getting reliable feedback can be hard. Most people who play your initial prototype will tell you it's great, and most of the time they are lying to be polite.

You shouldn't judge your playtesters too harshly, though. They will want to encourage you. Being creative is hard and many people are sympathetic to anyone who attempts it. The problem is that you want players to be honest with you about their experience so you can better evaluate if your game is providing the correct one.

Even the players who are honest often don't understand how to help you improve. They might make suggestions that don't fit your vision or even get frustrated and give feedback that, to put it kindly, isn't exactly constructive.

As a designer, you should be aware that *what* people tell you with words is unreliable. The good news is that *how* and *why* people tell you things is almost always reliable. This is where video feedback comes in.

Asking your testers to record themselves playing will allow you to watch them play through specific events and their corresponding reactions. Their facial expressions and vocal intonations won't sugar coat their feelings about what is happening.

If you are unsure about how to get video recordings of playtesters, here are a few ideas.

First, seek out friends or family. Ask a close friend or family member to play your prototype on your computer or laptop. You can use Open Broadcaster Software (OBS) and a webcam to record a quick video of them playing. If you don't have a webcam, make sure you at least record audio.

Second, consider contacting streamers. Live streaming on Twitch and YouTube is very accessible these days. You may find that you have friends or

family who have started streaming as a hobby. Streamers are an excellent source of video feedback. They usually have setups that make it easy for them to make a quick recording.

Lastly, don't forget about other indies. Many indie scenes have local meetups and times allocated for sharing and playing each other's games. Find your local meetups and ask other developers to try your prototype.

Before you get feedback, ask your playtesters to talk through their thoughts as they are playing. Resist the temptation to explain your game to them. If you are present and they ask you a question, answer with a question of your own. "What is your first guess?" is a general purpose question that works in most situations.

As you evaluate the video footage, learn to interpret it and dig deeper. Ask yourself why they might be reacting in that way. What are the positives you see? Does the game provoke a strong response? Do the players seem engaged or are they simply going through the motions? Where do players get confused or frustrated?

When testers try your prototype for the first time, they are building a mental model or "map" of how the game works. That map may include knowledge like "if I press the A button, I jump" or "if I land on an enemy's head, it will die." Players engage with the game and experiment with it in order to build this map. Is the mental map they are building accurate? How can the game give better feedback so that their map gets accurately updated? These are all questions you should be thinking about as you review the video.

The Playtesting Partner

The easiest way to get frequent feedback is to playtest the game yourself. Obviously, this solution comes with flaws (your own biases) but it's still a decent option.

The next best option is to get a *playtesting partner*. A playtesting partner is a single individual who is willing to record themselves playing short prototypes every week or so.

Message a trusted friend and ask them if they will volunteer for the job. Let them know that the favor will include playtesting little prototypes and that they should record themselves. These playtests will generally be short (five to ten minutes) so it won't be a big time commitment.

Once you've established the expectation of the role, ask them to playtest a small prototype for you seven days from now.

Getting a playtesting partner will allow for more frequent feedback that is also more objective than your own internal feedback.

As you hone in on a prototype that is working well, you'll start to move from a single outside playtester, to a small group of playtesters. Keep the group relatively small during pre-production. I'd recommend around five to ten playtesters. You'll bump up this number once you start alpha testing in production.

Conclusion

A kind feedback environment is essential if you want to iterate and improve upon your designs.

The gamedev feedback environment is a tough environment to learn in. The wicked elements make feedback unreliable. Luckily, you can take steps to change the environment and make it kinder. Working closely with playtesters, getting feedback through video, and finding a playtesting partner can alter the nature of your feedback, making it more reliable.

As you move your feedback environment from a wicked one to a kind one, you'll more fully understand the problems in your design. This understanding

will lead you to stronger solutions—solutions that solve the problems your playtesters experience while staying true to your vision as a designer.

Chapter Summary

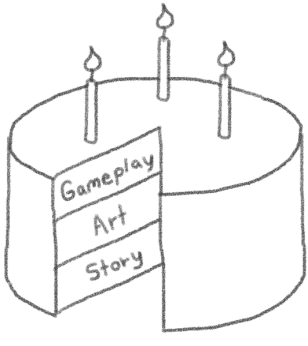
- In a wicked learning environment, experience and feedback aren't useful. We should attempt to create kind learning environments.
- Players often make suggestions about how to change the game. It's difficult to work backward from these suggestions and derive their experience.
- Working closely with playtesters allows you to dig deeper and better understand the problems they are experiencing.
- Recording players gives you more reliable feedback.
- Finding a playtesting partner can increase the frequency of the feedback you get.

5

Make a Vertical Slice

The Multi-Layered Cake

Imagine a multi-layered cake. This cake represents your game, and each layer of the cake is a different element of your completed game. As an example, you may have a gameplay layer, an art layer, and a story layer.



Each of these layers is a crucial contributor to the flavor of the cake. You want to experience that flavor, but you don't want to have to eat the entire cake just to find out what it tastes like. Luckily, you can simply cut out a single slice. This slice will carve through every layer and you can taste the cake without having to eat it all.

Creating a vertical slice (or demo) is a common approach in game development. It removes uncertainty about several different aspects of the game before you commit to moving into production.

The first three actions for pre-production—rapid prototyping, following the fun, and watching your playtesters—explore and remove uncertainties around your game's systems and mechanics. Making a vertical slice is the final action of pre-production. It explores and removes uncertainties around the other areas of your game's design, like the art, music, sound effects, story, and user interface (UI). It's possible you have explored some of these areas already, and that is okay. But it's time to make sure you have them locked in.

Until you have a finished vertical slice, you won't be able to tell if you are at the potential point or the production point.

Greg Donovan, one of the creators behind the *Saint's Row* series, describes the vertical slice as a "gate to production" in his GDC talk titled "The Vertical Slice Challenge."[1] It's a simple metaphor, but an important one to understand. If you don't have a strong vertical slice, you aren't ready for production.

Uncertainties to Address

Let's look at the uncertainties you need to address in order to complete the vertical slice. I can't make a list that includes every uncertainty. Each game will have its own unique list. I can, however, include common uncertainties:

- Will the art capture the interest of my target audience?
- Will the music and sounds enhance the experience or distract from it?
- Will the theme or story support the game mechanics?

1. https://youtu.be/1h4M7zDPg1Y

- Will the UI be easy to navigate and understand?

Again, this is not a comprehensive list. This list gives a starting point. For some games, this might be enough. For others, you may need to add your own uncertainties.

Over the next few sections, I'll address each uncertainty with my own experience developing vertical slices.

Art and Visuals

Your game's art direction will likely have the most impact on the success of your game. Make sure you address this uncertainty.

For better or for worse, players will judge your game primarily on screenshots and video footage. Creating an art direction that captures the interest of your target audience will make your game much easier to sell.

If you are not an artist, consider hiring one. If you plan to spend money on the development of your game, this is the first place I would put it.

Keep in mind that games don't have to have realistic graphics in order to capture the interest of your target audience. The art direction you chose should reflect the kind of game you are making and the kinds of people you want to attract.

The fastest way to find the right art direction for your game is through mockups.

These mockups will give you an idea of your resolution, color palette, font, and the medium you will use. I personally use pixel art because I'm familiar with it. Finding the correct resolution is important with pixel art projects, especially ones that will have text as text can be difficult to fit into lower resolutions. These are uncertainties that need to be answered, and a mockup is a quick way to do that.

Start by taking inspiration from other games. Find a bunch of games with art styles you already like. Use them as a starting point for your mockup. Don't assume you have to start from scratch.

When you first start creating mockups, stay loose. You should be playful and not worry about how it looks. Create a wide variety of loose mockups. Don't commit to any of them just yet. Artwork has its own pre-production and production phases.

If you've hired an artist to create mockups, ask them not to commit to any one idea. Artists are already practiced at sketching out ideas without committing to them. Get several different mockups and evaluate them together. Look for ones that immediately tell your players what genre your game is, while also expressing what makes your game unique.

Once you find your favorite mockups, share them with a larger group of people to find the mockups that resonate best. Ask them what kind of a game they imagine the mockup is describing. Does it look fun to them?

The answers to these questions can let you know if you are ready to move on from creating mockups or if you still need to create more.

Once you have a mockup that describes the genre well and appeals to your audience, you can use that mockup as a template for creating the rest of the art for your demo.

Up until now, you've been using placeholder graphics for our prototype. Now, you can throw those old placeholder graphics out the window. Enjoy the satisfaction of watching your visuals come together to create a beautiful, cohesive look.

Music and Sounds

Music and sounds often get overlooked by new developers. I've fallen into this trap myself. However, having a strong soundtrack and satisfying sound effects is something players appreciate.

This is another place where spending money on a contractor might be wise. Hiring a musician and sound engineer can make a huge difference in the quality of your game, but I understand that not everyone will have that luxury.

For the art direction, we created loose mockups in order to test our ideas quickly. For music and sounds, it's important to do the same. Get several different musical test pieces and try them in your game.

For sound effects, you might be able to find quality sound effects to purchase in a bundle that fit your game. I've done this on my own projects—while also altering and mixing the sound effects to make them more unique.

Create a system in your code that makes it easy to swap out and try new sound effects or music. The faster you can swap it out the better.

On my projects I create a resource that has a list of every sound or song. I can quickly drag and drop different sound effects into the slots of that list—even while the game is running—and hear each sound in context. Quick iteration will be your best friend here, especially if you are new to this.

Sometimes a song or sound effect will sound great by itself, but then sound terrible when played in the context of the game. On the flip side, a sound effect or song might not feel right until you get it into the game. As an example, in *Tic-Tac-Tanks,* I remember using the sound of an umbrella opening as part of an explosion sound effect. You never know what sounds will match the different actions in your game until you try them. Try sounds and music you wouldn't normally consider. You might discover a song or sound that fits really well that you couldn't have anticipated.

Theme and Story

All games present some type of story to the player. How this story is presented is unique to each genre and game.

Even if the story isn't the main focus, try to gauge how well your players pick up on the story. Would making the narrative aspect a bigger part of the game engage the players even more? Will the game need some dialogue between characters? Would creating some background art create enough of a setting to express the narrative for your game without something like dialogue?

Vlambeer (creators of *Nuclear Throne* and many other popular indie titles) are known for creating detailed backstories and worlds for all of their games.[2] The backstories and world don't always make it into the final game, but they affect the design. They influence the sprite designs, song names, item descriptions, and background art. The story is woven into the game, even though it isn't directly stated with dialogue. Give this a try if you are building a game and don't feel like it has any story.

If your game is more focused on the story, watch the emotions of your playtesters at each story beat. Make sure the emotions they are feeling are the ones that you intend for that beat. Do you want them to laugh? Worry? Cry? Capture video or watch their faces in person. Consider ways you can change the dialogue or characters to create the story experience that you want.

UI and Menus

Having a menu and some basic game options is an important part of your vertical slice. There may be issues in your code that you won't even be aware of until you allow the player to change the game's resolution or volume levels.

---

2. https://www.gdcvault.com/play/1016430/Sensible-Nonsense-On-Fiction-in

We talked about resolution a bit with the art, but it plays a major role in your UI as well. Will your players be able to adjust the screen size? Does your game support widescreen monitors? What solutions will you be applying for different aspect ratios? These are all questions you should have answers to before moving to production.

Use other games—especially ones in your target genre—as examples for what options you should consider.

If you are struggling with UI layouts for your game you might consider exploring the Game UI Database.[3] It's full of screenshots highlighting different UI solutions in hundreds of different games and genres. It's a great place to look for inspiration.

Conclusion

Getting a taste of every part of your game before you leave pre-production will give you the information you need to evaluate your game idea for production. The vertical slice is the tool for this job. Focus on removing any uncertainties like art, music, sounds, story, and UI until the game no longer feels like an idea, but a small game. Once you've removed these uncertainties, you'll be ready to look at your game and determine if you have reached the production point, or if you are still at the potential point.

We'll look at how you can evaluate your vertical slice in more detail in the next chapter.

Chapter Summary

- A vertical slice is a demo that "cuts" through every aspect of your game

3. https://www.gameuidatabase.com/

(art, music, sounds, story, and UI).

- The purpose of the vertical slice is to remove any remaining uncertainties before production.
- Players will judge your game from screenshots and video. Make sure you make a strong first impression.
- Don't neglect music and sounds. Find ways to test different songs and sounds quickly.
- Explore the story of your game, even if the story isn't the focus.
- Create some basic menus and UI elements. This may be harder than you think it will be.

6

The Production Point

The Danger of Potential

"Potential" is a scary word. If something has potential, it *could* be great but *isn't* yet. The gap between those two realities is the scary part.

In the **Introduction** of this book I established the two points at which you can commit to a game: the potential point and the production point.

In the chapters since I've introduced you to the two phases of development and the four actions of pre-production: rapid prototyping, following the fun, watching playtesters, and making a vertical slice. Now it's time to evaluate your game for production.

Are you at the potential point or the production point?

When my brother and I released the first vertical slice for *Demonlocke,* we were still at the potential point, but we committed to the project anyway. Progress immediately stopped and this ended up costing us wasted time, effort, and money.

However, the story doesn't end there. At some point during *Demonlocke's* development I was contacted by an indie developer who goes by Miziziziz on YouTube and asked if I would be part of a four developer game jam collaboration video.

For the collaboration, each developer would make a game in a single month using a shared art pack and document our process through video. Then, Miziziziz would compile all the videos together into one and upload it to YouTube.

I accepted the challenge and started prototyping a new game. My idea was to create a mashup of tic-tac-toe and some 3D tank assets included in the jam.

As I worked on this tank prototype, I couldn't help but compare it to *Demonlocke*. Progress on this new game was quick and I was having a lot of fun. My motivation levels were high and the players who tested the prototype liked it. Most of the feedback was for polish or more content, which didn't require me to rework entire mechanics and systems.

After a month, I had a mostly finished game. It still lacked polish and content but it was feature complete. The systems were all working well and players wanted more.

Demonlocke was nowhere near that point. The systems and mechanics were still all over the place. Even after years of development, playtesters were confused and disinterested.

I could tell right away that the tank game prototype was at the production point. I committed to working on it part time with another developer and we finished and released it about a year later.

This contrasted experience between *Demonlocke* and *Tic-Tac-Tanks* led me to the most important ideas in this book. It showed me first hand what the difference between the potential point and the production point looked like.

Let's consider some key differences between the two projects I was working on:

	***Demonlocke* Prototype**	***Tic Tac Tanks* Prototype**
Systems and mechanics	Uncertain	Established
Polish	Lots of early polish	Used free online filler assets
Playtester feedback	Bored and disinterested	Interested, wanting more
My motivation	All time low	High
Path to finished game	Uncertain	Polish and add more content

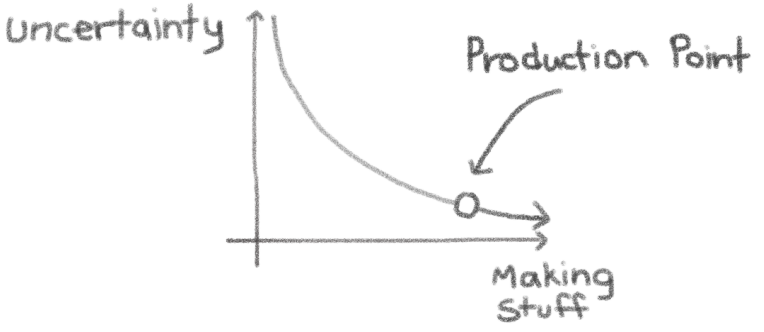
There are a number of different features we could compare. In my own experience as a developer, the features in the table are the ones I tend to focus on. Solving the systems and mechanics of your game early is a big one. It gives you a path forward. The end of the game is in sight.

This table is a simplified version of a complex idea. It's not always going to tell you if your game is ready for production, but it's a start. The next tool at your disposal is called the *uncertainty curve*. This curve can give you additional information to use while evaluating your vertical slice.

The Uncertainty Curve

The main goal of pre-production is to reduce the uncertainty in your project as quickly as possible. This uncertainty is reduced by doing; not thinking or planning. That is why I recommend specific actions during pre-production.

The uncertainty in your game can be loosely graphed on a curve. Let's take a look at what this curve looks like:



If you had to draw a point on this curve, where would your project fall? As you consider this question, consider what doubts you have about your game:

- Are all the systems of the game in place and working?
- Do playtesters enjoy the game?
- How long do they play it for?
- Are people immediately interested in this game from the screenshots?
- Do you have a clear path to the end of this game?

If the answer is "no" to any of these questions, you must return to pre-production and either consider what action you can take quickly to generate a quantitative answer to these questions or explore new prototypes.

You can't answer these questions by planning. You must answer them with action.

After getting the feedback on *Demonlocke*, I made the false assumption that if I made a plan for solving the design issues in the game I could then jump into production. I was wrong. I should have returned to pre-production and tested my plan before ever moving to production. You *make* and *play* your way through systems and mechanics and *plan* your way through content.

If after considering these questions you still think you are ready for production, there is one final uncertainty to address before moving on. That final uncertainty is funding.

Funding

A great time to secure funding is during the transition from pre-production to production. Securing funding at this point answers one of the most important uncertainties: will people pay for your game?

There are many ways to secure funding for your project, but I'm going to split them up into five main categories: self-funding, crowdfunding, publisher funding, grant funding, and early access funding. I'll add in mixed funding to the end as a bonus.

Self-Funding

For self-funding, either you, family, or close friends have the funding to support you as you work on your game. Maybe you have sufficient savings, or maybe you have a significant other that is working and willing to support you as you pursue this project. Maybe you have a rich uncle with a tower filled with gold that he swims in and is willing to donate some of this money to help you create your game.

The best part about self-funding is that you don't have to contact publishers or create and run a crowdfunding campaign. It doesn't require much work. You can jump right into production and focus your efforts on the game. One of the main drawbacks of self-funding is that it can be harder to know how interested people are in your project, or if people are willing to fork over their cash to buy it.

Crowdfunding

Crowdfunding has been a popular method for securing funding for quite some time now, and many small teams and studios have used it to great effect. Popular crowdfunding platforms include Kickstarter, Indiegogo, and Patreon.

Patreon works as an ongoing campaign (like a subscription service) where customers pay each month to support you as you work on the game. In return, you provide monthly perks (such as behind the scenes devlogs, or early builds of the game) as a way to compensate them.

Kickstarter and Indiegogo are both one-time campaign services where you create a campaign (usually for thirty days) and your customers pledge specific amounts of money to support the project. In return, you promise specific perks like copies of the game, early builds, merch, exclusive calls with the team, artbooks, etc.

I personally have found more success through Kickstarter, whereas I didn't like my experience as much with Indiegogo or Patreon. However, I know other developers who have found great success on all of these platforms. If you decide to crowdfund your project, pick the platform that makes the most sense for your situation. If it doesn't work out, you can always try a different platform or funding method.

One of the benefits of crowdfunding is that you can get an immediate feel for how interested people are in your project. It answers the question of *"will people pay money for this game?"* I love this benefit because I've had people tell me they love a project I'm working on, only to find out when the project launches that nobody is actually interested enough to buy it. It can be hard to tell the difference between *"I really like this"* and *"I'd actually pay for this."*

Crowdfunding also has its drawbacks, though. Creating and running a thirty-day campaign can take months of preparation and hours and hours of work

during the campaign. Anyone who has run a crowdfunding campaign before knows how much work goes into it.

Even after successfully funding your game, managing a campaign can be a lot of work. You will feel pressured to deliver a great experience to your backers. That pressure can be a weight on your shoulders sometimes as you work. Be extra careful of stretch goals, they are scope creep and can contain a lot of uncertainty that could delay your release.

Some backers won't be kind when they give you feedback on your campaign and progress. You are dealing with a crowd and that can be hard. You should try to be understanding of their position and expectations. Do your best to under-promise and over-deliver.

Waiting until the production point before crowdfunding can mitigate a lot of these issues and reduce the amount of stress you feel during your campaign. You won't be worried about major uncertainties in your project after receiving funds from backers because you'll have addressed these uncertainties already during pre-production. You can move right into production and give your backers frequent updates on the content you are adding. You may still run into issues, but you will have a schedule moving into production. You'll be able to see the light at the end of the tunnel, and that relieves a lot of pressure.

Publisher Funding

Publisher funding is another great way to secure funding for development.

Publishers are great if you are mainly interested in making games and not at all interested in marketing or building a brand. Publishers can handle the marketing side for you and allow you the freedom to simply make games.

Waiting until the production point to secure a publisher has benefits as well. Publishers intuitively understand the concept of uncertainty. The more uncertainty there is around a project, the more potential risk is associated with

it. If you have a great demo and video of your playtesters enjoying that demo, you will have a solid case to pitch to a potential publisher. This will increase your chances of getting a yes from them.

Each publishing deal is going to be different, and you should reach out to other indie developers to get a second opinion before ever accepting a publishing deal. Be wary of new or relatively unknown publishers. There are a lot of shady publishers and publishing deals. Many indie developers have been cheated or taken advantage of. Research your publishers well. It's better to self publish or crowdfund a game than accept a bad deal with a shady publisher.

Some publishers I would be willing to work with personally include Humble Bundle Publishing, Chucklefish, Devolver Digital, Indie Fund, Paradox Interactive, and Double Fine Presents. This is not a comprehensive list and there are other great publishers out there to consider as well.

Grant Funding

Grant funding usually won't get all the funding you need for your project, but it is a great way to supplement your funding. Sometimes there will be grants available for indie developers from your government or other organizations. You should always be on the lookout for these because they can be a great way to secure some extra funding and usually don't have a lot of strings attached.

Early Access Funding

Another popular funding method is early access. Early access allows players to purchase your game before it is finished in order to help fund the continued development of it.

In my opinion, this works best for games that have high replay value and require a lot of content: games like roguelikes, deckbuilders, and high score

games are examples of this. If your game can already be played from start to finish and has high replay value but could benefit from additional content, it may be a good fit.

Players have come to expect a certain level of quality and commitment from early access, and rightfully so. Sadly, there are horror stories where developers launch a game into early access only for it to be a buggy mess and later abandon the project before ever completing it, leaving supporters high and dry.

Understanding this initial level of quality and the expected commitment is a crucial first step in a successful early access campaign. I wouldn't recommend moving to early access with a game that is still in pre-production.

Early access can have a lot of benefits when done well. Players are able to support a game and a developer they like while giving them feedback. This helps the game improve and funds the creation of additional content. The players get that additional content for free and can have a say in what gets added and how it is balanced. This ideal balance can be difficult to find, but if found, it can be a positive experience for both the developer and the players.

However, if that balance is not found, early access can become a curse. Imagine launching a $10 game into early access on Steam and getting five hundred sales. That might sound like a nice amount of money, but after Steam takes their percentage and taxes your $5,000 is going to be more like $3,000.

$3,000 dollars to make a game (assuming a standard indie game scope) is not a lot of money, but now you are committed to those five hundred players. They have just purchased your game and are expecting frequent content updates.

Hopefully your game will continue to sell during early access, and if it does you should be fine. However, there is a possibility that it stops selling and you still have an obligation to those five hundred players. It can put you in a really awkward situation as a developer: one in which you have players to support, but not enough money to support them.

If you consider Kickstarter as an alternative, requesting $20,000 for a campaign forces you to raise at least that much money. If you only raise $5,000, the campaign will be unsuccessful and you won't receive anything. That might seem like a negative, but it can actually be a positive. If $5,000 isn't enough to build your game, you don't want to have a commitment to a large group of backers that you can't deliver on.

Early access can work for certain projects and is a great tool for improving your game while raising funds to support continued development, but make sure you have considered the potential downsides before you jump right in.

Mixed Funding

We've talked about the five main categories for funding, but you don't have to pick just one. Many indie developers will mix and match methods in order to get enough funding for their projects. This is a great way to raise more funding than you could manage with just one method.

The order in which you secure funding can matter as well. It may be easier to get a publishing deal (and you might get a much better deal) if you have already secured funding and proven the marketability of your project through a crowdfunding campaign. Maybe you successfully raise funding in a crowdfunding campaign and also supplement those funds with your own savings and some grant funding.

Mixing your funding options is a powerful way to make sure you get the money you need to carry your project all the way through production and to launch day.

Conclusion

It's time to evaluate your game and either greenlight it for production, return to pre-production, or cancel the project completely. If you've done your job correctly in pre-production, canceling at this point should be rare.

If the game is ready for production, you can start securing funding. If you decide to fund through crowdfunding or a publisher, you'll want to hold off on production until either the campaign is fully funded or you sign with the publisher. If you self-fund, you can jump right in and get to work.

Making it to production is a huge milestone. Congratulations! The road through pre-production can be a difficult journey. There are so many twists and turns. The road through production can be long and uphill but if you've done your job right in pre-production, it won't have as many surprises.

Chapter Summary

- Don't move to production at the potential point. Make sure you've reached the production point.
- Does your game still have major uncertainties? If it does, move back to pre-production and take action to remove them.
- The final uncertainty that needs to be addressed before production is securing funding.
- Consider different funding sources like self-funding, crowdfunding, publishers, grants, early access, and mixed funding.

Production

"The second phase of the project during which you build the game." – Mark Cerny

7

Create and Maintain a Routine

Fence Posts

There are exactly twenty-nine fence posts in my backyard.

I know this because a few years ago I put in a vinyl fence.

I'd never put up a fence before and I wanted to do it well. My family came to help dig the post holes and I rented a post digger to make it a bit easier. We managed to get all the holes dug on a Saturday. After the post holes were finished, I purchased the vinyl posts, gate kits, panels, and quick setting cement.

However, I still needed to cement the posts in and I knew I wouldn't be able to do it all in a single day. I needed a routine.

I got my brother to help me and we decided to do two posts each morning.

First, we'd grab a few bags of cement from my grandma's trailer and a few posts from my garage. Next, we'd mix the cement in a wheelbarrow. Then, we'd put the post in the hole and make sure it was straight and lined up with the other posts. Finally, we'd pour the cement around the post, double check that it was still straight, and let the cement dry.

This process took about an hour each morning. Breaking the whole job down into these morning chunks made it easier for me to do.

Now that I have put in a fence myself, I have a better understanding of how much work it is.

I recently drove out to visit my sister in North Dakota. As we transitioned from the tall majesty of the Rocky Mountain ranges to the calm beauty of the midwestern plains, I couldn't help but notice the miles and miles of fence posts. Each post raced past me as we drove nearly 70 mph on the freeway. It would have been impossible to count them all.

The amount of work that went into making so many fences was mind boggling to me. I have a deep respect for the farmers and ranchers who must have dedicated each day to consistently digging and placing fence posts.

Like finishing a fence, finishing a game is a lot of work. Whether your game is a small backyard fence or miles and miles of fence on the midwestern plains, If you try to do it all at once, you will fail. You need to break it up into smaller chunks.

During pre-production you want to explore, jumping around different ideas until you find something that works. But during production, it's important to have a routine of consistent progress, like the morning fence post routine I had with my brother. Doing a manageable amount of work each day builds momentum that can carry you through the difficult times and get your game to the finish line.

My second commercial game, *Tic-Tac-Tanks*, was a puzzle game at its core. This put me in a bit of a dilemma when it came time to create the levels for the game. I wasn't that keen on designing puzzles.

However, I knew the game was fun and I wanted to make sure that there was enough content to release it as a full game, as opposed to a small demo.

After moving to production, I created a routine that I could easily maintain over a few months in order to create all the puzzles for the game.

My routine involved creating two or three puzzles each morning, and I focused on making consistent progress. Even if some of the puzzles turned out to be duds, I knew I could throw them out or rework them later.

Tic-Tac-Tanks released with fifty different levels, each with multiple solutions. It takes the average player about four to five hours to complete. My target playtime for the game was three hours, so I managed to exceed that.

Creating so many different levels was difficult for me. I've personally struggled with consistency my whole life. As a creative person, I tend to be more whimsical in my approach to life and scheduling. Over the years I've gained a better understanding of my own personality and how to work with it. I've found ways to achieve enough consistency to finish a game.

There are four main systems that help me to achieve this level of consistency:

- Putting First Things First (do it as early as I can in the day)
- Minimums (build small habits)
- Keeping the Streak (track those habits)
- Habit Pairing (combine two or more habits)

Over the next few sections I'll cover each of these in more detail.

Putting First Things First

I can always tell where my priorities are by looking at the first thing I do when I get free time.

The flip side of this is that I'm able to make a task a bigger priority simply by scheduling it earlier in my day.

I schedule gamedev as early as I can in my day. For me, this means first thing in the morning—after writing. However, everyone's schedule is different, and you might not be able to do it first thing in the morning. That's okay. Find

a method and time that works for you but stays true to this principle. Make gamedev a priority.

Minimums

There are two types of friction in physics: *static* friction and *kinetic* friction.[1] Static friction is the model used when an object is at rest. Kinetic friction is the model used when an object is in motion. Interestingly, a larger force is required to overcome static friction. In other words, it requires more force to start moving an object than to continue moving it.

This difficulty to start is something I can relate with. I must usually be an object at rest.

I recently started the "game a week challenge" described by Bennett Foddy in **Chapter 1**. I've been anticipating this for months, and yet, I still find myself avoiding gamedev.

It baffles me that my brain will work so hard to avoid doing something I so desperately want to do. I've even found myself taking out the trash, shoveling snow from the driveway, or even doing finances all to avoid making games.

This isn't always the case, though. When I'm working on a game and in a flow state, I can do it for hours without any desire to stop. But starting, and then finding, that flow state requires so much effort.

To solve this problem, I keep my gamedev habits small. The smaller they are, the less effort I have to exert to start.

Instead of "work on a game for thirty minutes each day," which is a significant task for many indie developers, my habit might look like "turn on some music and run my game." This makes it both easy to keep my momentum on

1. This is one model for friction called dry friction, but there are others as well: https://en.wikipedia.org/wiki/Friction

the hard days, and easy to get started on the good days. This minimum idea isn't a trick, either. If I turn on music and run the game and then decide I don't want to work on my game, I won't. Most of the time I do end up wanting to work, though. I've already done the hardest part: starting.

Turning on some music and running my game was the exact habit I used while working on *Demonlocke,* and I was able to maintain and enjoy a constant routine with the game for months after starting production. Unfortunately, the game wasn't ready for production at the time, so much of the work I did was wasted. This was the consequence of moving to production at the potential point.

Keeping the Streak

"Yard by yard it's hard; but inch by inch, anything's a cinch." - Unknown

One of the most powerful aspects of the fence post routine at the beginning of the chapter was my daily streak. My brother and I made sure to set new posts every single weekday.

Keeping streaks like this has become such a normal part of my life that it hardly seems notable. On a day to day basis, it doesn't feel significant, but looking back I can see how much it has affected my life for the better.

I use a habit tracking app on my phone called Loop Habit Tracker.[2] It's simple, adless, and free. Each day it reminds me of my habits and I can check them off once completed.

I have several habits listed right now. Some have longer streaks than others. My habit of brushing and flossing my teeth each night currently has an active

2. https://play.google.com/store/apps/details?id=org.isoron.uhabits&hl=en_US&gl=US&pli=1

streak of 2,263 days. That's over six years where I haven't missed a single day—including days away from home like camping trips and vacations.

This book wouldn't exist without my habit tracking practice. At the time of writing this, I have an active writing streak of 797 days.

We all have hard days. We can't expect to be in a good mood or healthy all the time. Work within the limits of your own body and be patient with yourself. Use minimums so that keeping your streak doesn't put too much pressure on yourself. In the book *Atomic Habits,* James Clear recommends never missing two days in a row. Missing a single day is a slip up; missing two days is the start of a new habit.

Habit Pairing

Habit pairing involves taking an existing habit and pairing it with a brand new habit. The new habit is easier to form because it piggybacks off of the established habit.

When I first started writing this book, I was struggling to get into a routine in the mornings with writing. However, every morning my partner makes me hot chocolate (I'm lucky to have her) so I decided to pair my existing hot chocolate habit with writing. Now, I wake up, get my hot chocolate, and start writing. The two habits are linked together. Pairing them together immediately fixed my morning writing routine.

I've since been able to write consistently and output the words needed for this book.

Conclusion

As you create a strong routine—by putting first things first, using minimums, keeping the streak, and habit pairing—you'll make consistent progress on your

game. This routine will give you the industrious mentality necessary for production.

Make sure that the routine you establish is sending you in the right direction. Having a routine for a vinyl fence is a waste if the fence goes off in the wrong direction. My routine while finishing *Tic-Tac-Tanks* was designed to help me create the different puzzles in the game. That was the direction I needed to go in and I made sure my routine would get me there.

It's wise to find genres that require content you are already interested in making, but there are always going to be parts of production that aren't glamorous or exciting. Routines are the key to making your way through those parts.

Chapter Summary

- Finishing a game is a monumental task. Having a consistent routine will be a powerful ally as you work towards that goal.
- Prioritize your game by working on it early in your day (in the morning if possible).
- Build a routine around small tasks that get you started.
- Use a habit tracker to stay consistent with your goals. Don't lose your streak.
- Pair your habits with other enjoyable and established habits.
- Make sure your habits and routines are leading you in the right direction

8

Create a Schedule

A Game Without a Plan

Back in 2017 I started working on my first commercial game: *Endhall*. I worked on the game part time and my pre-production phase took about a year. At that time I didn't understand the pre-production/production split, and it ended up hurting the project later in development.

At the end of pre-production I thought I was ready to release the game. I'd added a final boss and a simple story. The game had randomly generated levels and different unlockable skills each time you played. I thought this would add enough replay value to the game to justify a $5 price tag. Unfortunately, I was wrong.

I sent the game to playtesters and the feedback was primarily positive. There were a few playtesters who mentioned that the game felt small, but this group was a minority and they didn't seem too upset. In retrospect, I should have paid closer attention to them.

I released *Endhall* on Steam and was disappointed by its reception. Because the game took about a year of part time work, I was hoping to make around $20,000 in revenue, but the game ended up making about $7,000—before taxes and Steam's cut.

Endhall's main flaw (according to reviewers, and I agree) is that it lacks content. I attribute this flaw to a production cycle that ended prematurely. The core idea of the game is fun and works well, but it doesn't last and leaves players unsatisfied when the credits roll.

Despite its flaws, *Endhall* currently has an 84% positive review score on Steam with a hundred reviews. It's still a fun game that many players still enjoy and I'm proud of it.

However, I failed to fully develop the game's content during production. It impacted my review scores and likely impacted my sales. I should have decided on a target playtime and used that to plan out more content. I shouldn't have skipped the production phase.

In this chapter I'm going to teach you how to set your production phase up for success. You'll create a schedule to plan out the content for your game. This schedule will prevent your game from suffering *Endhall's* fate.

The Schedule

Creating a schedule is the most important action at the beginning of production. For all you planners that had a hard time with the freeform nature of pre-production, now it's your time to shine.

Here are the tasks you should focus on when scheduling out production:

1. Setting a target playtime.

2. Planning how to scale your game.

3. Creating a calendar and setting deadlines.

Each of these tasks will give you a clear roadmap to finishing and releasing your game.

Setting a Target Playtime

Now that you're in production, you must consider the target playtime for your game. This number will affect the rest of your schedule. Decide on the best length (usually in hours of playtime) and what content you need to get it there.

Start by considering how much content and playtime your game already has. Hopefully, you'll have an idea of how long it takes to play your demo. If you are unsure, ask your testers how long it takes them to beat it. If they recorded videos for you during pre-production, you'll already have some data points from those videos, but it's still important to know if they are playing your demo outside of that footage. It's possible they are putting more time into the game than you realize.

Whether you want a forty-hour experience or a four-hour experience is entirely up to you as the designer. I personally like shorter experiences, but I know a lot of gamers see longer experiences as being a better value for their hard-earned money. Consider your audience when deciding how long to make your game. Players who like RPGs are often concerned with the length of the main story. Players who like roguelikes are more concerned with the replay value. Look at other games in your genre to get a feel for the player base and what they expect for content.

Once you've got a feel for what players expect for the genre your game is in, make a list of all the content in your current demo. How long did it take you to create that content? Focus on the content itself. How much playtime does that content provide for your players? Once you have these numbers, use them to extrapolate the time cost for the rest of the content you need. Write down these estimates.

Let's say your demo currently provides two hours of playtime and generating that content took about four months. We can estimate two months for every hour of content.

Let's also suppose you are aiming for eight hours of playtime and your demo provides two. You will need twelve more months of development just to reach your beta. These numbers are estimates, but having a baseline from real data is the best starting point.

Content output is a common way to increase playtime, but it's not the only way. You might consider creating more playtime through replayability. This is a reliable and proven way for indies to reduce the amount of content they need to create, while also providing more value to their customers.

Spelunky is my favorite indie game of all time. The game can be beaten from start to finish in a matter of minutes. I've personally beaten the game in under eight minutes. However, I have logged over one hundred sixty three hours of total playtime in the game on Steam. I have a massive amount of playtime because the game has high replay value. I'm not the only one replaying this game either. Video Game Insights,[1] a useful website for estimating different statistics on Steam games, estimates the average Steam user playtime to be around one hundred twenty seven hours—a number only slightly lower than mine.

Nintendo uses replayability to give more value to customers as well. *Pokemon,* for example, lets you pick one of three starting companions at the beginning of the game. That choice drastically affects your decisions as a player for the rest of the game. If you choose Charmander, it doesn't make much sense to add a Ponyta to your team. Both monsters are fire types, and having a variety of types on your team is a wise strategy. Choosing a specific class at the start of the game can increase the replayability of that game by the number of classes. The three starters at the beginning of *Pokemon* might easily triple the playtime for many players.

Setting a target playtime is crucial if you want to release a game that doesn't feel like a demo. Create estimates about how much content you need and look

1. https://vginsights.com/game/239350

for ways to add replayability. These strategies will help you reach your target playtime and provide as much value to your players as possible.

Planning How to Scale Your Game

It's important to understand how to scale your game from a demo to a finished game during production. Some games are harder to scale than others. If you are making an action adventure game, you will probably scale the game by adding more levels to explore, more treasure to find, more items to collect, and new enemies and bosses to fight. Scaling games like these is a relatively straightforward idea. However, not all games are so straightforward.

Let's consider my two commercial releases.

I had an easy time scaling *Tic-Tac-Tanks*. It was a puzzle game so I was able to add more levels to it. However, the levels started to feel the same. As a solution, we introduced unit modifiers to allow for some variance in later levels. For example, we had an armor modifier to give units the ability to tank hits and a double action modifier that allowed units to act twice. These simple modifiers allowed me to create new levels and more complex puzzles. Without them, I wouldn't have been able to scale the game up to the four-hour target playtime that I was aiming for.

I had a much harder time scaling *Endall,* which contributed to my too-short production cycle. Because the game was a roguelike, replayability was the most logical tool for adding playtime. However, adding new levels and skills didn't seem to give the game more replayability. I suspect this came from the difficulties I had around the power curve in the game.

Endhall has a logical difficulty curve. The enemies increase in number and in strength as you progress. Unfortunately, the power curve doesn't complement the difficulty curve. Your character becomes more powerful after every level no matter what skill you choose. The different skills don't actually affect the power

curve. I believe replay value could have been created if the choices of the player directly affected the character's power level. Wise and informed choices would lead to more power, while poor choices would lead to less power and eventual defeat.

I did get the choices during battle right in *Endhall*. Each choice you make in battle is meaningful and will contribute to winning or losing that battle. The micro choices matter but not the macro choices. The game has tactics but not strategy.

Understanding this could have given me the tools I needed to scale the game. I'm certain if I had scaled it well it would have gotten much better reviews and likely more sales.

How you scale each game will be different. At its core, it's going to involve a balance between the difficulty curve and the power curve. Look to other similar games in your genre for ideas if you are unsure of how to do it, but make sure you don't skip it.

Creating Your Calendar and Setting Deadlines

To create a calendar, I use Trello. Trello is a simple, free online tool for planning and organizing projects. I use it during both pre-production and production, though it is more important during production.

Trello is simple at its core. It is made up of boards, lists, and cards. Boards are where you keep your lists. Cards can be stacked on different lists, and cards can have deadlines and labels. That's about it: boards, lists, cards, deadlines, and labels. There are a few other features but those are all extras and don't contribute to the core experience of the tool.

Trello allows for a large amount of lists, but I usually try to keep my boards simple. The simpler I keep them, the more I actually use them. I've found that larger boards get too complicated for me to get value from them.

In their simplest form, my boards will have three lists:

- Todo
- Working
- Completed

Once I have these main lists, I create a card for each task in production and place it in the proper list. All tasks start in the "Todo" list. Each morning, I move the tasks I want to work on that day from "Todo" to "Working." Once I finish the task, I mark it as completed and move it to the "Completed" list.

Many people use the label feature as a way to estimate how long a task will take. I don't like this method as I believe any long task should be broken down into smaller tasks anyway. For my own projects, I create labels for my game's engaging actions and make sure that each feature I add supports those engaging actions. If a feature I have planned can't be labeled under one of my game's engaging actions, I remove it. This system helps me to evaluate each feature as I'm planning and make sure that it supports the core actions of my game.

Trello has a variety of addons you can install to your board. They call these add-ons "power-ups." The first power-up I always add is the calendar power-up. This makes it easier to set deadlines and milestones. I strongly recommend you use this power-up as well.

As my project progresses, I add a few new lists to my board. The core lists will always be "Todo," "Working," and "Completed," though.

Some other lists I like to add are:

- Priority Todo (if "Todo" starts to get too long)
- Random Ideas
- Visual References

- Player Feedback
- Feedback to Reevaluate
- Design Thoughts
- Programming Ideas

These lists are placed to the right of my core lists.

How you set up your Trello board is up to you. You can copy my board as a starting point, but you don't have to follow it exactly. Experiment to find a system that you like. The point of Trello is to help you scope and stay productive. Any list or card you create should serve at least one of those purposes. If your board is getting too complicated and it is preventing you from being as productive, try something simpler.

Defining the exact content scope for your project in Trello can take a lot of work, but it will be worth it to get an idea of how close you are to finishing.

Once you've created your calendar in Trello or a similar program, you will be ready to set the deadline on each card. The purpose of these deadlines is to push you, but not to overwhelm you or stress you out.

Be patient with yourself at first. If you set a deadline that is too hard or impossible to reach, change it. Try to make reasonable adjustments that continue to stretch you but not stress you.

Getting a perfect deadline is impossible. But with some experience, you will get better at it.

Conclusion

Starting production without a plan makes it difficult to finish your game. Even if you do manage to finish it, it may lack the content and polish to give

your players sufficient value. *Endhall's* reviews and, likely, sales suffered from a too-short production cycle. Don't fall into this trap.

Make sure you start production by creating a schedule. This schedule should include setting a target playtime, planning how to scale your game, creating a calendar, and setting deadlines. These tasks will give you the roadmap you need to make it through production and deliver a high quality game to your players.

Chapter Summary

- Scheduling is the first action you take in production to start refining your game for release.
- When creating your schedule, focus on these main tasks: setting a target playtime, planning how to scale your game, and creating your calendar and setting deadlines.
- Calculate the target content amount using the content you already have and your target playtime.
- Consider replayability strategies to help you hit your target playtime goal.
- Make sure you have a plan for scaling the experience of your game. Adding more content doesn't always scale the experience.
- Use Trello or a similar project management tool to write out every aspect of your schedule.
- Set a deadline for each task in your Trello board. Stretch yourself but don't stress yourself.

9

Run an Alpha and a Beta

Cobblestones

I grew up in rural Utah in the backyard of the majestic Rocky Mountain ranges. My parents built a small log home on a lot near my grandparents' land. A large creek cuts across that land near my home. Every spring, the creek floods with runoff water as the snow melts on the mountain tops. At night, during this flooding season, you can hear the rumbling of millions of cobblestones as they roll down the creek from the mountains above.

These cobblestones begin their journey high on the mountains. They are rough and jagged. As they roll down the mountain, the rough edges are slowly worn away. The rocks collide with each other billions of times on the way down the mountain until they are all round and soft.

It's still amazing to me when I walk out to visit the creek and see the thousands upon thousands of rounded, polished stones.

At the start of production, your game is like a stone at the top of the mountain. It still has those jagged edges and hasn't been refined and polished. As you run the alpha and beta builds, you polish away those rough edges, making your game appealing to a larger audience.

The steps I recommend while running the alpha and beta are:

- Filter the content.
- Address bugs.
- Implement quality of life features.
- Consider accessibility.
- Add localization.

We'll cover each of these steps throughout the rest of the chapter.

Filter the Content

During the alpha, you should concern yourself with filtering the content you have already added to the game.

Identifying what content to keep and what content to cut will give you a cohesive vision for the final game. Don't be scared to remove content. The game is just as much about the content you take out as it is about the content you leave in. Before you filter your content, make sure that content has variety. Having variety will provide more options as you locate the best stuff and eliminate the worst.

When you are ready to start filtering your content, use your alpha testers. Create a form with specific questions that direct the type of feedback you get. The form will have questions about the content of the game. This constrains the feedback given and directs your testers to the parts of the game you care most about during the alpha. To get the best of both worlds, you can also have open-ended questions near the end to allow your testers to express themselves openly and break free from your feedback constraints.

Here are some questions you might consider for your form in an alpha test:

- What was your favorite piece of content (character, enemy, level) in the

game?

- What was your least favorite piece of content (character, enemy, level) in the game?
- What was your favorite action (spell, skill, attack) in the game?
- What was your least favorite action (spell, skill, attack) in the game?
- What aspects of the game felt well-balanced?
- What aspects of the game felt unbalanced?
- If you could add more to this game, what would it be?
- Is there anything else you would like to tell me?

These questions allow you to filter your content and still leave room for more open-ended feedback near the end.

As you get responses to these questions, look for recurring themes. Don't make major changes to any part of the game unless you feel it needs changing or you see a pattern in the feedback you get.

In the next section I'll give you some ideas for questions to consider for your form during your beta.

Address Bugs

During the beta, you should take a deliberate and systematic approach to locating and solving bugs. Removing as many bugs from your game as possible is your main focus during this build.

In order to find as many bugs as possible, you'll need to have a large number of testers with a variety of different testing devices.

Make sure you get testers from all your launch target platforms (for example, Windows, Mac, and Linux). This might seem obvious, but while working on *Tic-Tac-Tanks* I made the mistake of only getting testers for Windows and Linux—not Mac. I did this because I was able to get away with it for my first game, *Endhall*, which launched on all three platforms with no issues even though I never tested on Mac! However, with *Tic-Tac-Tanks*, it didn't work out the same way. On launch day, we were forced to remove Mac as a launch target because we were unable to find a way to get the game running on Mac. It was an especially embarrassing mistake, even accounting for my experience with *Endhall*.

Once you have testers for all target platforms, choose a method for your users to report bugs. The easier it is for them to report a bug, the more likely they are to actually do it. You might look into software that is specifically designed to handle bug reporting like Jira, Monday, or Log Rocket. I personally use Trello and Google Forms because they are easy to use and I understand them.

Setting up a bug reporting system where players are encouraged to give detailed and clear reports will save you time and effort when it comes time to fix bugs. If you are lucky enough to have a video of when the bug occurred, you can simply watch the footage to get an understanding of how the bug is triggered. However, this won't always be the case. Many of your playtesters won't be able to record themselves during the alpha and beta. You are at the mercy of their memory and reporting skills. You want to set up a system that guides your playtester through the reporting process—encouraging them to create a detailed report.

Here are some questions you might consider in your form:

- What platform/OS were you testing the game on?
- What hardware (specifications) were you testing the game on?
- What are the steps to recreate the bug (critical)?

- Do you have a screenshot or video of the bug?
- Did you see any error messages? What were they?
- Is there any other information you'd like to share that you think could be relevant to finding and fixing this bug?
- How might I contact you later if I have follow up questions (email, Discord, Twitter)?

Using questions like these, going from specific questions to more open-ended ones, will allow you to gather crucial information while also allowing the playtester to use their own intuition and share information that might be relevant.

Some playtesters may already be in direct contact with you via a Discord server or a Slack group. Getting bugs from these testers can be a simple conversation, but make sure you still log these bugs. Keeping a record of when bugs were reported and when they were fixed can be useful later if the bugs pop up again.

For Trello I create three lists and log bugs in those lists. The lists are titled the following:

- Logged Bugs
- Bugs in Testing
- Fixed Bugs

Any new bug found ends up in "Logged Bugs." Once it is fixed, I move it to the "Bugs in Testing" list. It's common to "fix" a bug, only to later realize that it wasn't completely solved. Having bugs in the "Bugs in Testing" list for a while can help you to be sure the bug is fixed for good. There is no sure fire way to know when a bug is ready for the "Fixed Bugs" list, though. I just move it there

when I feel it's been in testing for long enough that I'm no longer worried about it.

The final step in the process is prioritizing your bugs. You might consider using the label feature in Trello to prioritize your bugs. For example, I use these different labels:

- Game-breaking
- Immersion-breaking
- Slightly Distracting
- Frequent
- Infrequent
- Rare

Label each bug with the labels from this list that apply to it. A frequent, game-breaking bug is going to be at the top of your priority list, whereas an infrequent, slightly distracting bug will be lower.

Prioritizing bugs is important because you'll never have the time to fix them all. Focus on the big ones that negatively affect the player experience. Some minor bugs can give your game character, and speedrunners love them.

Implement Quality of Life Features

As your testers play your game, they are going to have small grievances with certain parts of it.

Sometimes these grievances are easy to fix. In these cases, you should implement a solution right away. Solutions that only take a few minutes to implement can often drastically improve the experience for many players. These quality of

life fixes are a bargain. They cost you hardly any time and provide a lot of extra value to the player. Keep an eye out for these.

For example, platform games have a small quality of life feature commonly referred to as a "coyote jump." Imagine a coyote in a classic cartoon, hanging for a split second in the air after leaving a cliff. This feature allows the player to jump for a few frames after leaving a ledge and assists players if they slightly mistime a jump. Mistiming a jump from a ledge can feel terrible because it seems like the game is ignoring your inputs. A coyote jump can alleviate this frustration for players and create a game that feels responsive.

However, there are also suggestions that would take more work. For these, you should make a list. Watch for points of strong frustration. If your players are getting frustrated with parts of the game that shouldn't be difficult, like the controls, that's a red flag.

I implemented a quality of life feature for *Endhall* that allowed players to use hotkeys for different skills. I had a player request this because they didn't want to have to use the mouse to select a skill every time they wanted to use it. This feature was tricky for me because I was new to Godot Engine and didn't know how to implement hotkeys. I figured it out, though. The game is better for it and the player was grateful.

Small quality of life fixes shouldn't be ignored because they can make a huge difference in how nice your game feels to play. They add a level of polish that can help your game stand out from the rest.

Consider Accessibility

When making a game, it's important to consider accessibility. How large is your game's audience? This is a question you must answer for yourself. Nobody else can answer it for you. You may want to target a specific audience. I know developers who make games for family members and loved ones as gifts. I've

even seen developers make games to use in wedding proposals. In cases like these, you only need to consider a small audience. On the other hand, you may want your game to reach as many people as possible. If you are attempting to reach a broader audience, you must consider how accessible your game is.

Video game accessibility has been a mainstream topic in game development for a while now. Going into every specific accessibility feature is outside the scope of this book. However, there is a comprehensive free resource online called Game Accessibility Guidelines[1] that gives you all the information you need to get started.

The site is split up into three sections: basic, intermediate, and advanced. Each section gives different considerations and guidelines. The difficulty of implementation starts off easy in the basic section, and then increases in difficulty as you move from intermediate to advanced. The basic section has a list of relatively easy changes you can make that will make your game accessible to a much wider audience. On the other side of the spectrum, the advanced section contains complex changes that target fairly specific audiences.

Start with the basic tab. Look through each consideration and evaluate whether it applies to your game. Come up with plans to implement the ones that do. I think you'll find that many of the suggestions are already familiar to you and relatively simple to add.

Another solid resource for video game accessibility is Mark Brown's "Video Game Accessibility" playlist on YouTube.[2] This wonderful series covers topics like audio visualizers, colorblind palettes, customizable controls, and optional assist modes. Brown also covers popular games that handle accessibility well, and ones that could be improved.

---

1. www.gameaccessibilityguidelines.com

2. https://youtube.com/playlist?list=PLc38fcMFcV_vvWOhMDriBlVocTZ8mKQzR

As you consider different accessibility options for your game, you will open your project up to a much wider audience. An audience that will be grateful to the extra effort you put in to include them.

Add Localization

Once your game is nearing completion, you should consider localization. Localization is the process of making your game accessible to other cultures and languages.

Getting your game's text translated to other languages can be done in a few different ways. You could pay a localization company to do it for you, or you might get help from your community of players. Many indies have done the latter. The drawback of paying a company is the cost, but you will get more consistent translations and guaranteed results. If paying a company isn't an option, consider the community. Many players are willing to help get the game translated into their native language. If only some parts of your game have been translated, be sure to let potential customers know what areas have been localized before they commit to buying. Bad translations and localizations are frustrating for players and will often result in negative reviews.

Many developers associate localization with simply translating a game into other languages, and that is a large part of it, but it may involve more. Consider any graphics or cultural references that may not apply to other audiences. How could you change them to be inclusive of those audiences and their cultures?

In the Disney Pixar movie *Inside Out,* there is a scene where a character reads a sign. As the character reads, they point to each word. In the English version of the movie, the character is animated pointing from left to right. However, not only did Pixar translate the sign for other languages, they also changed the animation of the character to point from right to left for languages that read

that way. These are the types of considerations you may need to make when localizing your projects.

Many modern game engines have built-in tools for localization. The process involves unique keys in the code that get replaced with text or images depending on the selected language. The data for these keys is usually stored in a comma separated value (CSV) file. You can generate these files from a spreadsheet. Fully localizing your game is a big job that requires a lot of work, but setting up a simple spreadsheet for your localization in Google Sheets and sharing that with your community is an easy first step to get you started.

Here is an example of a basic localization spreadsheet:

	en-us	pt-br	es-mx
CONTINUE	Continue	Continuar	Continuar
SETTINGS	Settings	Configurações	Opciones

On the top row you have the regional language codes. En-us is American English, pt-br is Brazilian Portuguese, and es-mx is Mexican Spanish. There are many different codes for different regions. Look them up online for a comprehensive list.

On the left column you have the key. This key is what you will place in the code. When it comes time to display text, your engine will use the code in combination with the key to retrieve the correct text. It's kind of like playing *Battleship*.

How this table connects to your game's code will be slightly different depending on your engine, but it's similar across the board. You can use the above table as a starting point and check the documentation of your particular engine for more information on implementing it.

Conclusion

Feedback during production should focus on polishing your game to remove all the rough edges. Alpha testing can help you identify the type of content that meshes well with your game and Beta testing can help you locate and fix any bugs. As you get closer to finishing, you should also address quality of life features, look at accessibility, and add localization. These actions will prepare your game for the deliverable in production: the release.

Chapter Summary

- The purpose of feedback during production should be to polish your game.
- How you communicate with your playtesters matters. Use forms to direct their feedback.
- Create systems for locating, tracking, and fixing bugs. Use forms and Trello to assist you.
- Keep an eye out for quality of life features you can implement to improve your player experience.
- Consider how many people you want your game to reach. If you are targeting a large number of people, make sure to consider different accessibility options to open your game up to a wider audience.
- Localizing your game into other languages is another way to widen its potential audience.

10

Follow a Launch Plan

Stress at Launch

It was launch day for *Endhall*. I had players in Discord chat and on Twitter asking when the game was going to be released, but I still had what felt like a hundred tasks to complete. I needed to upload my trailer, add screenshots to Steam, and fill out the game's description. Time was ticking along with or without me. I was stressed.

Hours before my launch, a Discord user messaged me with a game-breaking bug. The game would completely freeze. After a bit more testing, we discovered that it happened to every single player. *Endhall* was unplayable. My stress levels skyrocketed.

How did I end up in this situation? I ran alpha and beta tests during production. There were hardly any bugs and I'd fixed all the bugs we'd found.

Unfortunately, I hadn't followed a launch plan. I tried to wing it. It's cliche, but the common phrase "if you fail to plan, you plan to fail" especially applies to launching a game. Having a clear plan with specific steps for launching *Endhall* could have saved me from this unfortunate situation.

Launching a game is complicated. You could easily fill an entire book on this one aspect of game development. I'm going to give you an essential plan for launching your first game.

Let's take a look at the launch plan:

1. Announce a release date (at least a month ahead of time).
2. Make a press kit.
3. Create a launch trailer (with a call to action).
4. Build a wishlist or an email list.
5. Create a release candidate.
6. Test the release candidate.
7. Wait a week.
8. Launch your game.

Each of these steps will create a better launch experience for you. Over the next few sections I'll cover each of these steps in more detail.

Announce the Release Date

Make the launch of your game an event. Your launch is already going to be special for you, but you want it to be special for your customers as well. Announcing a launch date at least one month in advance—preferably through a trailer—can create anticipation for your game's release.

It is crucial to build anticipation if you want customers to be excited about your game. Have you ever experienced the phenomenon where the anticipation of an event is more enjoyable than the event itself? Maybe like a planned vaca-

tion and where the weeks leading up to it were more enjoyable than the vacation itself? Setting a launch date can create this same feeling in your customers.

There is a goldilocks zone for the duration between your announcement and your release. You want enough time to build anticipation and get the word out, but not so much time that the hype around your game dies down. The balance here is difficult to find and depends on a lot of factors. For beginners, I'd recommend shorter durations between your announcement and the release—start with about a month. It's easy to get lost in the flood of AAA announcements and big releases if you are relatively unknown and you announce your game's release too early.

Make a Press Kit

Having a press kit, even if it's a simple one, is an important step to take before launching your game. A press kit is a collection of text, images, music, video footage, and trailers that you share with press or content creators. This collection allows them to quickly and easily write articles or make videos about your game and its release. When you provide a press kit, the likelihood that your game will be covered by press or content creators goes up because it's easier to cover and share.

I made the mistake of not making a press kit for my first game, *Endhall*. That's a mistake I don't want to repeat. *Endhall* got some coverage on YouTube and a short article from a Linux gaming news site, but not much else. Writing this now I feel a bit bad that I didn't take the time to create a simple press kit. It would have made that article easier to write and potentially opened up my game to other writers who just didn't want to put in the work to track down gameplay footage or screenshots. I'm grateful to the author of that article and to YouTubers who gave my game a shot on their channel.

A press kit can be as simple as a shareable zip folder hosted somewhere online, or it can be as fancy as a complete website. You'll probably want to scale the size of your press kit relative to the level of interest you have received in your game. If you have a known publisher or you successfully funded the game in a Kickstarter campaign, consider making a website. It will look more professional and you already know there is interest in your game.

What you put in your press kit is up to you. I recommend you start by highlighting the hook of your game. Your hook is the combination of the genre and the unexpected twist that sets it apart from other games in that genre. I've found two useful ways to phrase your hook. The first is the "but"[1] method, which I learned from Jonas Tyroller, a popular YouTuber and game designer. The second is the "unanswered question" method. The one you use depends on your game and its unique hook.

Let's take a look at both methods and a few examples for each of them.

The "but" method looks like this: A standard [blank] but [blank].

Here are some examples of the "but" method:

- *Shovel Knight*: A standard medieval platformer *but* your hero uses a shovel instead of a sword.
- *Spelunky*: A standard platformer *but* the levels are different every time you play.

The "unanswered question" method starts with questions like these:

- What would happen if you [blank]?
- What would you get if you [blank]?
- What would it look like if you [blank]?

1. https://youtu.be/x46VdUtvDDs

Here are some examples of unanswered questions:

- *Slay the Spire*: What would happen if you mixed a roguelike with a deckbuilding game?

- *Crypt of the Necrodancer*: What would you get if you mixed a roguelike with a rhythm game?

These two templates are good starting points for understanding what is expected and what is unexpected about your game. However, they are only starting points.[2] Find ways to change the wording or phrasing them so it doesn't get repetitive.

For my second commercial release, *Tic-Tac-Tanks,* I wrote my hook like this: "*Tic-Tac-Tanks* is a tactical coffee-break puzzle game inspired by tic-tac-toe."

The genre is: "coffee-break puzzle game."

The twist is: "tanks meet tic-tac-toe."

I hope these templates give you a starting point for your phrasing that you can remix and expand on.

Once you've written your hook for your press kit, you are ready to gather some screenshots and video clips.

Use your hook as a guide for your screenshots and video footage. You want to highlight the hook in any materials you provide. It's the most interesting part of your game and will probably make the best articles.

You should provide a lot of variety. If you give variety, the press can select images or videos that other articles haven't used yet. For bigger news sites or content creators, you may even consider sending them exclusive images or video and let them know it's exclusive.

---

2. For a deeper dive into this concept, check out the book Impossible to Ignore by Carmen Simon, PhD

Once you have a press kit, you might wonder how to share it. You could start by contacting press sites directly. You might also include a link to it in the description of your announcement trailer. If you crowdfund your game, you can also include a link to your press kit on your crowdfunding campaign. The key principle here is to make it easy for press to find your press kit once they have found your game.

Don't skip your press kit. The easier it is to share information about your game, the more likely people will do it. Make it easy and focus on the hook of your game. This will increase your chances of getting press through articles or video and guide the conversation to what makes your game special.

Create a Launch Trailer

I'm going to start off this section by recommending you pay a professional to create a trailer for your game. In my opinion, there are two areas where your money is best spent when making an indie game. Those two areas are your game's artwork, and your game's trailer.

Creating a compelling trailer is like playing a difficult song on the piano. It's obvious to everyone watching how much experience you have.

With that disclaimer out of the way, I understand that paying for a professional trailer isn't going to be an option for everyone. I made both trailers for my commercial games.

Endhall's trailer was weak but I learned a lot and I think the trailer for my second game, *Tic-Tac-Tanks,* turned out much better. However, it's still obvious when watching that neither of them are professionally done.

In this section I'll give you some questions that you can use as an outline while making your trailer. These questions come from Derek Lieu's wonderful email newsletter on making trailers for games: in particular, an email titled "Trailer Structure via Audience Questions." You can sign up for his email list

and learn more about making trailers through his articles and courses.[3] I highly recommend checking them out.

First, let's start with what information your trailer should prioritize. Your viewers and potential customers are going to have some questions about your game and they expect your trailer to answer those questions. You could answer these questions with text on the screen, but that should be a last resort. Ideally you want to answer these questions with shots of the game itself.

Here are the questions we'll consider:

1. What is the game genre?
2. How is this game different?
3. Why is that interesting and/or noteworthy?
4. How does the game challenge me?
5. What is the scope & longevity of this game?
6. What is it called and how do I find it?

Notice that this list is ordered. In other words, you want to answer each of these questions in this specific order. Let's take a quick look at each question.

What is the game genre?

This question should be easy to answer for most games. You can include a few gameplay shots that show what genre the game falls under. These shots should establish the expected actions of that genre.

---

3. https://www.derek-lieu.com/

Remember those engaging actions from the pre-production? Those make up the core of your game and you should include footage of them early in your trailer. A lot of developers call these actions the "verbs" of the game because they are the things you do. In *Minecraft* these would be something like: explore, mine, craft, fight, and build.

Some games will have a lot of engaging actions and some will only have a few. Games that have only a few engaging actions usually have lots of variation in what those actions can do in a certain context. The *Mario* games have running and jumping, but Mario is able to jump in a variety of different ways. He can jump under a block to break it, or jump on an enemy to kill it. This creates diversity in the game, even though it doesn't have quite as many actions.

Think through your engaging actions and the different ways they can be used. Show these off in your trailer and let your players know what they get to do in your game. This will establish the basic gameplay and genre.

It's generally recommended that you turn off UI and HUD elements when recording trailer footage. However, you might consider leaving some UI elements in if it helps establish the genre for your players. If your game is a deck-builder, make sure you have some shots of the cards and the process of adding cards to your deck. If your game is a puzzle game, show off some of the puzzles and solutions. Let viewers know exactly what type of gameplay experience they should expect. This will establish a standard and nicely set up for the next question.

How is this game different?

It's time to show off the twist of your game. This question is harder to answer than the last one. If the first shots establish the standard, then these shots should establish the twist. If you are making a deck-building game, what about it is unique from other deck-builders? Is there a mechanic in your game that isn't

in others? Maybe you have a unique look or art style? Find the thing that makes your game different and show it off here.

Just adding a twist isn't enough, though. Players want to know how this twist influences the design of the rest of the game. This idea leads into the next question.

Why is that interesting or noteworthy?

Once you show off the twist, you need to show why it matters.

In these shots you'll want to show your viewers why they should care about the twist from your previous shots. Something can be different and not meaningfully different. We've all played those games with thousands of different items only to find that most of the items are nearly identical except for a few random changes in stats. A rusty bronze sword isn't that different from an old bronze sword. Make sure your game isn't a rusty bronze sword.

Find shots of your game that show how the twist changes other parts of the gameplay in big ways. If your deck-builder has a card fusion mechanic, then show how that might affect the battle with the final boss. Does it affect the choice players make when adding cards to their deck? The answer to this question is going to be specific to your game, but make sure you answer it.

How does the game challenge me?

Once you've established the genre, twist, and how that twist plays a meaningful role in your game's design, your viewers are going to be sold on the idea. But, they'll still want to make sure the game is going to challenge them. A game can have a cool idea and not be interesting to play if it doesn't challenge them in

some way. Trailers will often show the player dying or losing at this point. It gives an "all is lost"[4] feel to the trailer's arc.

Not all games are going to be hard. Maybe your game is literally a walk in a park. That's okay. However, all games challenge something in the player, even if the challenge isn't how quickly they can press a button. Maybe the game has some emotional challenge to it. Maybe it challenges a belief the player has. Maybe it challenges cultural norms. Find what it is about your game that challenges players and show it here.

What is the scope and longevity of this game?

By this point your players are excited to play the game. They have one last question they want answered before clicking the buy button, though. They want to know the scope of the game.

The answer players are expecting from this question is highly dependent on the genre. Be upfront with your audience about how much content and playtime your game has. You don't have to give the playtime number directly in the trailer, but you should hint at it through the shots you show here. Often trailers will show the character moving through several unique environments, fighting a variety of enemies, using different attacks and weapons. This helps establish how much content is in the game and what the player has to look forward to.

If your game is short, there is a specific audience for that as well. Make sure you set clear expectations for your viewers so they know how much playtime they are going to get.

---

4. The "all is lost" beat is a story beat used in many novels, plays, and movies. I recommend reading Save the Cat! Writes a Novel by Jessica Brody for more information on story beats and structure.

What is it called and how do I find it?

The last question to answer is the easiest. It is your call to action. Let the players know what the game is called and where they can buy it. The call to action can be as simple as: "Buy it now on Steam, GOG, or Itch.io."

You could even replace the text for each store with their logo, or list the logos below.

Hopefully your trailer has been compelling enough to convince them to pull out their wallet and click that buy button.

Some Traps to Avoid

When making a trailer there are some specific traps to avoid. Avoiding these traps will help you make better trailers and keep your viewers engaged and watching until the end.

- Avoid feature lists. Players care about what they can do (engaging actions) not how many items your game has.
- Remember to turn off your UI and your game's music while recording footage (keep sound effects on).
- Avoid long trailers. For small indie games, anything over two minutes is probably too long.
- Don't put your logo at the start of the trailer. If you have to do it, do it at the end.

I want to give you a starting point for making your own trailer, but at the end of the day, finding a way to raise funds for a trailer could make a huge difference in your game's sales. Again, I can't recommend Derek Lieu's email list enough.

It is full of valuable information for making great trailers. I wish I had paid for a professional trailer for my first game. Whatever you decide to do. Remember that first impressions matter when it comes to selling your game. Your trailer and game art are going to be the first impression people have of your game. Make them the best they can be.

Build a Wishlist or an Email List

Building a wishlist on Steam and an email list is the best way to ensure a strong launch.

Wishlists and email lists allow you to contact interested players immediately on the game's launch. Each person on your list is a potential customer.

Early in Steam's life, many devs promoted a ten-thousand wishlist number as a way to get front page visibility on launch. While this number isn't always going to guarantee that outcome, it's certainly going to give you a stronger launch, and it's a reasonable goal to aim for as an indie.

Email lists have some strengths that wishlists don't. A large email list can be used for the launch of multiple games, while wishlists will be game specific. You also have more control over your email list. I personally use both email lists and Steam wishlists.

The sooner you start building these lists, the better. The beginning of production is an excellent time to start, since you are committed to the project and have polished art and gameplay to entice prospective players to sign up.

Creating a Release Candidate

A release candidate is a "rehearsal" build. You should treat this build like it is the final release build, and it very well could be.

Publish this build on all your target platforms. Steam requires you to upload a launch build a few weeks in advance. When I went to release *Endhall* on Steam, I was forced to postpone my launch date because I was unaware of this requirement. Going through a rehearsal of *Endhall's* launch would have prevented me from encountering this issue. It would have also prevented all the stress and frustration on the actual release date because I would have encountered the critical bug weeks earlier.

Make sure you create a release candidate build before launch.

Test Your Release Candidate

Get the testers from the alpha and beta builds to test your release candidate. Address any serious bugs they find in this build, but ignore smaller issues. Those can be fixed after launch. You want your launch to go as smoothly as possible, and fixing a small bug could inadvertently create a larger issue somewhere else in your code.

Have your testers play the build on your target platforms. Again, this is a rehearsal for the actual release. You are testing your launch, not just your game.

Wait a Week

Wait a week before you publish your game. Give yourself a week where you don't work on the game, except for bug fixes here and there, and you focus on getting the word out. Email YouTubers. Connect with the press. Focus on other areas of your launch, but don't focus on the game. Once you launch, you'll be glad you took this small break from the code. This is your calm before the storm. It's a moment to recharge and prepare mentally before launching.

After launch, it will be a mad rush to fix bugs and support your customers. Take advantage of this short break to get some rest.

Launch Your Game

After going through each step of the launch plan, you can calmly and confidently press that launch button, knowing you are ready and rested for what comes after.

Publishing a game on Steam is one of the scariest things I've ever done. The publish button is the hardest button to push. It's a big moment. It's also your moment. Enjoy it.

Conclusion

Following a simple launch plan can make launching your game a much more enjoyable experience. Take the time to make a plan. The final deliverable of production is the release of your game. A launch plan will help you get there.

Use a release date to build anticipation for your release. Make a press kit to increase the chances of getting press coverage. Create a launch trailer that answers questions your audience may have about your game and gets them excited to buy it. Use Steam wishlists and an email list to ensure a strong initial launch. Create release candidates and test them before you release your build. Wait a week before launching.

If you do all these steps, you'll feel better prepared when it comes time to push that launch button. You've worked diligently to get here. Be proud.

Chapter Summary

- Launching a game without a launch plan is a sure way to run into issues. Create and use a launch plan.
- Announce the release date for your game at least one month in ad-

vance.

- Make a press kit to make sharing news about your launch easier.
- Create a launch trailer to build hype for the launch and get the word out.
- Build wishlists and an email list to increase your chances of a strong day-one launch.
- Create a release candidate build for your game (this should be a potential final release).
- Test your release candidate and fix any serious bugs (ignoring smaller issues).
- Wait a week before launching.
- Launch your game.

11

Supporting Your Customers

Support and Reviews

Right after launching your game, your focus should shift from making the game to supporting your customers. Your job now is to listen to and understand their concerns and do your best to address them. Be polite and courteous. Assume good intentions.

Negative reviews or hate towards your game can be hard to handle. As a small team or solo developer, you aren't shielded from criticism. There is no large corporation to take the blame for you. You may be blamed for flaws in the game and it's not always easy to deal with that. However, I do have some ideas that may help.

If you receive negative reviews, consider that the authors of said reviews may be close to loving your game. Elie Wiesel famously said that "the opposite of love is not hate, it's indifference."[1]

The strong feelings in negative reviews indicate that the reviewer likes your game quite a bit, but certain parts ruin the overall experience for them. Seek to understand what those areas are and evaluate your ability to address them. If

---

1. In U.S. News and World Report 27 October 1986

you feel yourself taking a review personally, take a few days to work through your feelings. Consider writing a physical letter response. This letter should contain your response completely unfiltered. Once you have written it, you can burn it or throw it away.

Let's look at an example of a negative review that *Endhall* got:

> "Do you wish more demos charged you money to play them? If so, this is the game for you! This isn't a full game, this is a paid demo. It is very well designed while it lasts, but it's very simple and has no replayability. Once you've played it once you've seen every enemy, ability, and idea. When I first beat the boss and the credits rolled, I said out loud 'are you ******* serious?' It's a shame, because it's a very good start. If the developer actually finished the game, I would highly recommend this. As it stands, this is more like a tease than anything resembling a playable game."

Ouch! This review stung when I first read it. In *Endhall's* defense, this reviewer also gave *Hyper Light Drifter* a negative review, so they have high standards.

I can imagine the snarky start to the review came from a place of frustration. They paid money for this game and were enjoying it, only to have it end prematurely.

In the second paragraph, they mentioned good design and a strong start. They honestly liked what the game had so far.

I can look at this review now and learn from it. I truly believe that my too-short production phase for *Endhall* is what caused this issue. I appreciate this review for helping me learn that lesson, even if I don't agree with the tone.

To balance out the universe, I'll share a positive review as well:

> "*Endhall* delivers a coffee-break tactical brainteaser for less than what you would pay for the coffee itself. Roguelike genre veterans may find *Endhall* a little too easy—I beat the game on my second and third attempts—but casual puzzle gamers will find a good introduction to the joys of procedurally-generated gauntlets here. Fans of *Hoplite* and *Into the Breach* will find another byte-sized nugget of what they love. The clean graphics and retro-synth soundtrack convinced me to play through one more time after that first victory, but there's not enough variety in the learned abilities to keep me coming back for many more runs down the hall. But I do hope to see more from this developer, whether in the form of expansions for *Endhall* or in other games down the line."

Your Priority When Reading Reviews

When reading reviews you should consider what your priority is.

If you read negative reviews and your priority is protecting your ego or the success of your game, you will get defensive and frustrated. Each negative review will be an attack on you or your game.

On the flip side, If you read negative reviews and your priority is understanding the experience of those customers and addressing them, you will be curious and understanding. In addition, your customers may be more likely to cut you some slack and apologize for their mistakes as you interact with them.

Having the right priority isn't something you can simply tell yourself to do. Your priority can shift in an instant. You could start reading reviews with the

right priority and swap to the wrong one, leaving frustrated and angry at your customers.

If you can feel yourself getting defensive or frustrated, take a break. Don't respond in the heat of the moment with something you will regret later.

Limit Your Interactions

There are horror stories of developers who get defensive on Steam and end up handling support and reviews poorly. I don't want to put too much blame on these developers because there is a weird power dynamic going on here. It's one thing to get defensive if one person points out an insecurity you have; it's another to get defensive if a hundred people do it on a public forum. So I can understand why some developers end up reacting defensively and making poor decisions.

However, part of our job as indies is to limit our exposure to that type of negativity. If your goal is to understand, a couple reviews should be enough. You don't need to read every single review and all the comments—especially not the comments!

I can only process so much negativity in a day. If the reviews are only slightly negative, I can do a dozen or so. But if I stumble onto strongly worded reviews that target specific insecurities I have, I can only handle a couple. My mood influences my tolerance as well. If I'm having a bad day already, it's not a wise time to try to learn from negative reviews.

Be patient with yourself. Give yourself time to process information that hurts.

Some developers avoid reviews completely. However, if you are planning on making updates or adding DLC to your game, reading reviews is probably wise. Understanding your customers and their concerns will be a crucial part of understanding where to take your game post launch.

Once you are able to understand your customers, you can start to explore creative ways to support them and update your game. Understanding them means understanding their problems. It's okay if you don't always agree with their suggested solutions. Find solutions that fit your game's vision and test those solutions in updates to make sure they correctly address the issues you've identified.

Updates and DLC

Updates and DLC are a "bonus" production phase.

If launch goes well, adding more content or providing DLC can be an economic way to capitalize on your existing success. Double down. Maybe your game will do well enough that you can provide this content free and lengthen the tail end of your sales curve. People already like your game, why not give them more of it?

It's important to note that games can receive negative reviews initially and still sell extremely well. *No Man's Sky* is an example of this. It's a space crafting/mining game with infinitely generated planets and galaxies. The game built up a lot of hype during development and sold extremely well at launch. However, the majority of reviews were negative. Despite the lackluster reviews, the game still made a lot of money. The developers took this opportunity to double down on the relative financial success, and published a slew of free updates for the game. It has continued to sell well and the reviews have climbed with each new update.

If your game's sales page converts well, regardless of negative reviews, you should consider the route *No Man's Sky* took as an alternative to rolling the dice again with another game. Major content updates will keep your players coming back and bring in new customers.

Conclusion

After launching your game, make sure you take the time to support your customers. This support starts with understanding them. Make sure you have the right priority when interacting with them, and that you limit your exposure to negative reviews.

After you get an understanding of your customers, you can provide updates or DLC to support them and extend the life of your game.

Chapter Summary

- After launch, your job is to support your customers.
- The priority you have while reading reviews will affect how well you handle them. Prioritize understanding the experience of your customers over your game's success or your own ego.
- Limit your exposure to negative reviews.
- Post-launch content and DLC are like "bonus" production phases. They allow you to double down on a successful launch and extend the life of your game.

Conclusion

The Right Moment

You've come a long way since the beginning of this book. I hope you have a better understanding of the process behind making a game and the moments that got you here.

Despite what the title of this book may imply to some readers, the creation of a game doesn't rely on just one moment. It's a string of moments: millions of them. Each moment comes together to create the final collective creation. If making games depended on a single moment, I think a lot more people would finish them.

However, while gamedev doesn't depend on a single moment, some moments are particularly important. Some moments *define* all the moments that come after. These moments should not be ignored. Decisions during these moments should be made with calculated care.

The production point is one such moment. It's your chance to seize everything you want as an indie developer. Identify it and seize it deliberately.

Don't fall into the trap of the potential point. It's a wolf in sheep's clothing. Make sure that when you seize the moment, it's at the right time. Use the actions and steps I gave you throughout this book to avoid seizing that moment too soon.

The Actions Bring You to the Finish Line

As you develop your game, you won't always be conscious of the different actions of each phase. But as you run into obstacles, you'll reflect on and use each action to overcome them.

Starting in pre-production, maybe you'll feel yourself slipping into an industrious mentality, and you'll have to make several rapid prototypes to bring back a playful mentality. Maybe you'll be too focused on content, and you'll need to focus on systems and mechanics by finding the fun. Maybe you won't know exactly which prototype to pick, and you'll have to watch your playtesters. Maybe you feel the urge to jump to production too soon, and you'll instead make a vertical slice.

Then, in production, maybe you'll find it difficult to have an industrious mentality, and you'll need to create a daily routine. Maybe you'll struggle to focus on the content and you'll need to make a long term plan and schedule. Maybe your game will be buggy and inaccessible (Gasp! Never!) and you'll need to run alpha and beta tests. Maybe you'll be worried about having a strong launch, and you'll need to follow a launch checklist.

At each step of creating your game, these actions will help you maintain the right focus and mentality: the focus and mentality that will help you discover your best game and carry it to the finish line.

As long as you keep the two phases separate, you'll be able to modify this simple recipe and cook up your own masterpiece. Doing so will build your skills and confidence. With each game you release, you'll get better and better at each step. Finishing a game won't seem impossible anymore. It will still be hard—it will always be hard—but you'll have the knowledge and tools to do it.

Thank You

I'm grateful that you have taken time from your busy life to read this book. I hope that you found value in it and that the information in it will help you as it has helped me. Don't hesitate to reach out to me on Twitter (@uheartbeast) or through email (ben@heartgamedev.com) with any questions or comments. I'd love to hear from you, especially if you manage to use the actions in this book to release a game! I look forward to seeing the amazing creations you make.

Thanks,

Ben

About the Author

Benjamin Anderson—AKA Heartbeast—is a tutorial creator and indie game developer. He lives with his partner, three daughters, and son in rural Utah.

He loves playing *Super Smash Bros.*, hiking with family, reading, playing guitar, and spending time at the lake.

Ben started making games in GameMaker at the age of fifteen. He's been making small games ever since. You can purchase his two commercial indie games—*Endhall* and *Tic-Tac-Tanks*—on Itch.io, and *Endhall* can also be purchased on Steam.

In 2014 Ben started teaching other people to make games on his Heartbeast YouTube channel. The channel now has over 200,000 subscribers and hundreds of video tutorials for GameMaker, pixel art, and the Godot Game Engine. Ben also has paid courses on pixel art and Godot Engine that can be found on his website: www.heartgamedev.com. More than 75,000 indie game creators have leveled-up their gamedev skills using his online courses.

Acknowledgments

The idea for this book first started to form a few years back during a conversation with my brother Caleb. As we talked, I compared my experience working on *Demonlocke* with my experience working on *Tic-Tac-Tanks*. *Tic-Tac-Tanks* had a relatively smooth development experience while *Demonlocke's* was a total disaster. The development experience for these two games was so drastically different that I couldn't help but question why.

That question led me to an intuition that splitting development up into two phases—one for exploration and one for exploitation—was the critical difference between the two experiences. I made a YouTube video talking about these two phases and started the journey that led to this book.

When I first crowdfunded this book on IndieGoGo, I couldn't have foreseen all the work and support I would need to get it written and published. Self publishing this book has been harder than I ever could have foreseen. It went from a part-time project back in 2022 to a full-time project at the beginning of 2023. I wrote and rewrote this book multiple times during its creation. Early Access and Beta readers from the IndieGoGo campaign will know how far it's come.

There have been so many supporters by my side throughout this journey.

First is my partner Charly. She has been a constant support to me. She listens to my ramblings daily as I talk through my ideas and struggles. Taking an intuition and formalizing it into a book is a difficult thing to do and my conversations with her have been a huge help.

Next is my brother Caleb. He has also been a source of insight for me as I've developed this idea and book. My conversations with him were much less frequent, but helpful all the same. His curious questioning helped me explore the idea in ways I hadn't previously considered.

Next is my editor Isaac Welfare. Isaac was referred to me by Nathan Lovato (the founder of GDQuest) and he has been a wonderful editor to work with. He has the heart of a teacher and takes the extra time to explain errors in my grammar to help me avoid them in the future.

Next are the beta readers who donated their time and knowledge to give me feedback on the earlier drafts of this book. This group consisted of Jeremiah Cooper, Juraj Móza, Ibraheem Asad, Marko Zakrajšek, Jeremiah Dobes, Luke Johnson, Austin McKee, Chris Purvis, Vincent Loibl, Cody Wheat, Effie Kok, Jack Kinsey, and James Dunhour.

Last but not least are all the IndieGoGo backers who supported this book. Without them, it wouldn't be in your hands today. Their generous pledges gave me the opportunity to dedicate so much time to writing, drafting, and editing.

The highest pledge, at $215 USD, was Piotr Nowicki.

At $205 USD were Devin Kirchgassner, Mario Roberti, Preethi Vaidyanathan, and Kyle Szklenski.

At $75 USD were Nick Mckenzie, Matthieu Huvé, Alistair McFadyen, Ioannis Gialanskyi, Andre Schulz, Mark Vermulst, Joshua Jenner, Jon Warner, Francisco N. S. Cruz, Armin Becker, and David Lake.

At $65 USD were Garnett Gilchrest, Haddi Jazzaf, Matt Gardner, bandhippie, Jack Kinsey, Chad D Lines, Derek "Der999" West, Pennie Quinn, Shane Romeo, Derek Alexander West, James Jackson, Stevie Roder, Destin Piagentini, Daniel Alejandro Leiva Funes, Marshal Wong, Marcelino Florencio-Villa, Garrett Roberts, Stephen Rice, Steve Sebastian, Nikolaus Frier, Sylvia Rae Gallimore, Kayin Ilagan, Nathan Wing, Kris Loukas, Travis Woods, Roger

Moore, Marion Vanier, Patrick Turpin, pwj2012, Michael Harrington, and Ryan Glover.

At $50 USD were Samuli Leppänen, René Holy, Liam Kerr, Tom, Orestis Konstantinidis, Richard Mountain, Rémi Cornebise, Christer Gordon, Hans Hubregtse, Jason Hennequin, Alejandro, Manzanero Sobrado, Francesco Bagnoli, Magdalena Cielecka, Tom Kuper, Peter Stoklumd, Stanislav Khromov, Sebastian Christen Marote, Jamie Legg, Maisy Donachie, Kristhian Otero, Finn Davidson, kpandos, Onur, Pedro Henrique Ruschel, André Decher, Victor Eulálio, Simon Proctor, Zhiming Chen, Hanli Theron, Nicolas Counil, Casper Elshof, Martin Hohl, Bret Curan, Carsten Nikiel, Andre Luiz Ribeiro Simoes, Juraj Móza, Teo Lee, Christian Jäkl, Baptiste Quesnot, João Correia, Domingo Ortuzar, Joseba k Bilbao, Daniel Coleman, Neil Sheppard, Brendan McCarthy, Cathal O'Gorman, Bostjan Skamperle, David Kernaghan, rafa, Matthew Burgess, Michael Banzon, Kiril Berdichevsky, Alexandre López Noguera, Gianluca Magro, Jeffery Chiu, Samu Saari, Aga Acrobat, Mathijs Lemmers, Adrian Doan Kim Carames, Oliver Villar, Aaron Moore, Robert Larnach, Angel Iglesias, Thomas Crunelle, Dustin Vogel, Martin Karanitsch, Francesco Emanuel Faella, Denis Romanin, Gustavo De Micheli, James Cowlishaw, John Kelly, Samuel Ernst, and George Asiedu.

At $47 USD were David Krause, Grégoire André, Kolt Penny, Ben Flavell, Liam Swain, Kai, Amber Smith, Maciej Kozłowski, Kyle Abernethy, Jere Orava, Carol Beck, Emilie Carrier, Gustavo Pereira, Jessica Chisnall, Gibo Ryu, Drifter, Johan Planchon, Mitchell Coote, Alistair Hill, Gregor Kšan, Huang ChenTing, Marko Zakrajsek, Seng Tat Pan, Mark Berentsen, Alvaro Ariza Caliz, Isaac Mabillard, Roberto Abad Jiménez, Austin McKee, Michael Johnson, Mateusz Jakwoczyk, Liam Hughes, Pignouf3, Petr Tůma, Harryanto Harryanto, Jake Adams, André Kishimoto, Sixto Javier López-Triviño López, Ximo Planells Lerma, Dan Reeves, Peter Bjørn, Chris Shea, Marcel Zöller, Miguel Escobar Rojas, Lachlan Mitchell, Maximilian Bethke, Carlos Bruno da Cunha

Teixeira, Alexander Moore, Andris Liepins, Robert Tester, Panagiotis Xinos, Vuk Mihailovic Takimoto, Marko Savikurki, Daniele Vicinanzo, Dmytro Podgornyi, Mike Atkinson, Pietro Maggi, Elia Nolz, Tommaso D'Argenio, lxmcf20, Amar Šahinović, Ryan Nguyen, Andrejs, Sebastians Bicevskis, Afreytes, Ivan Zlobin, Dylan Moon, Jarmo Taskinen, Jorge Antunes, Michael Boyer, Matt V, Tim Engelhard, Fabrice Daniel, Thibaud Bousquet, Jo Zurasuta, Ivan Ho, David Amador, Tom Materne, Dean Bolte, Elias Jacobs, Olivier Wynyard Gonfond, Sloopysrock, Sean Bohan, Honk5000, Gusts92, S Arrowsmith, Shahar Butz, Colin Gourlay, Rémy Tauziac, Przemyslaw Hawel, Alexsander Gomes, Henry Elder, Jan Ramm, Lukas Kelemen, Adrian Perez Vales, Thiago van Dieten, Felipe Mello, Gordon Heger, Federico Agustin Rattay, Armindster, Abdulrahman Alshaya, and Jean-Pierre Widerhold.

At $40 USD were Braxton Anderson, Christopher Pereira, Ryan Shook, Seebreeze35, Ben Ament, Steven Splint, Austin Russell, Stephen Adams, Roderick Greening, Ned Mcl, William Stephens, Matthew Perdrisat, Joshua Tanguay, Deron Mann, Chris Jaroszko, Kayvon Ghoreshi, Peartree Games, Jeffrey Jabagat, Robyn Louise Giles, Ian Forcer, Sharyq Siddiqi, Jo Hanna, Francisco Carrera, Elijah Gale, Thatcher Peterson, Ryan Holt, Carter David Robert Charles, Henning, George Wyand, Brandon Navarole, Samuel Dost, Nicholas Allen, Bmoconno, Michael Smith, Alexander Mutuc, Lancia Jones, Davi Barbosa, Jeremy Sciarrino, Alex A, Ryan Arana, lnclaassen, Philip Ludington ,Emrys Kok, Andre Boulay, Ian Poma, David Dalisay. Kino A Rose, Beasticorn, David Snopek, Julia Couture, Ben "Aecium" Phipps, Steve Richards, Connor, Younglund, Alexander Dudek, morrisonman56, Shelby Cruver, Tyrel Souza, Samuel Sarette, Kevin Selwyn, Michael Kyle, Tim Klimpel, Karl Anderson, Alex Quinn, Wayne Wollesen, Isaac Chapman, Adam Detmar, Joanna Ballendorf, Matthew Douglas Ferguson, Dustin Poulisse, Brandon McHenry, Luis R Ramos, Brian Douglass, Cristian Morales, Levi Shaffer, Kyle Owen, John F Johnson II, Hudson Bielstein, Jonathan Poholarz, Chad Kosteroski, Michael

Baggott, Christopher Jagusch, Andreas Freiburg, Connor Linning, Hasan Odom, Caleb Anderson, Xn, Devon Johannesson, Joey Russo, Ateichelman, Patrick Coridan, and Lucas Magee.

At $37 USD were Dimitrii Jackson, Christopher M Sanyk, Brian Jackson, Ryan Miller, Bret Linne, Phil Ulrich, Michael McDaniel, Cody M, Shane Urbas, Everett Gregory, Benjamin Truman, Geoffrey A Bokuniewicz, Jordan Reed, Cody W, Logan Lang, Brayden Burrow, James Bollivar, Colton Croft, Ramiro Prado, Jason Carbonneau, Micheal Arsenault, Matthew Wexler, Jonathan Robertson, Kelvin Dealca, Tyler A Rizzo, Erik Onarheim, Nick Dariano, Ashley Mora, Lukecool, Benjamin Lanz, Kyle Hessel, Aidan Melendez, Diego Teran Rios, Jesse Craig Hughes, Daniel Villa, Sbr_rakin, Jeremiah Dobes, Ben, Ryan Mirch, Lucas Kelleher, Sarah Dickson, Greg Moss, Travis Johnsen, Bchick222, Nicholas Gingerella, Mnuvr, Adam J Belliveau, Jbitautas, Trevor Dean Bartlett, Philip John Basile, Greusser3 Eace, Jeremy Lugo, Matthew Santacroce, Caleb Wiberg, Moriel, Frederick Kramer Dal Pra, Tim Wellette, Matt Kalafut, Hayden Parthenay, Francisco Carrera, Kevin Lu, William Bottiglierie, Gregory N Bayles Jr, Matt Pfaff, Myles, Shawn Black, Christopher Mangum, Scott Foster, Victor Arriaza, Kyle Woomer, Terry R Cox, Josh Butner, Juvenal Esteban Magana, Lewis Allard, Michael Paron, Jonathan Ball, Cecilia Perhaps, Kevin James, Alex Borucki, Sean DeBuys, Matthew Walker, Jeremiah Cooper, Benny Romine, Kevin Batdorf, Christopher Maikisch, Kathleen N Sebree, B. A. Witt, Jacob Brabec, Samuel Davis, Geoff Petrie, Steven Hernandez, Drew Randy, Nicholas O'Sullivan, Ted Fitzgerald, Will Kourafas, Yang Pulse, Aaron Ramsey, Mike, Ezra Ouellette, Etienne Vallee, Anthony Delgado, John Riggles, Arthur Vinson, Craig Gardner, Nathan A. Hemmings, Gunnar Clovis, Dshift3, Kurt Loeffler, Hugh Bagan, Lep, Michael Policastro, Joseph McCormick, Norman Henson, Sarah Laurin, Eddie Hoyle, Andrew Smith, Cassidy Williams, Paolo Munoz, Brett Parker, Cameron Elless, Garrett Steffen, James Dunhour, Cole Andress, Moi-

ses Pelaez, John McDermott, Cameron Clausen, Veronica Canterbury, John Gibson, Jlquartet, Amy Eastment, Benjamin Ortiz, Jerimiah Gazlay, Josiah M Bradbury, Benjamin Arrington, Aaron Beshea, and Andrew Miller.

At $25 USD were Terence Chen, Jia Hao Woo, Sam.sjk, Dan Zaner, Alexander Schalk, Thomas Arnold, W. van Lit, Nic Reichelt, Tim H, Hanz Barker, Andrew Royer, Malik Ahmed, Nicholas Apicelli, Nycholas Martinez, Colby Autrey, Guilherme de Oliveira, Galen Swain, Lorenzo Massidda, Tenebrisdominus, Timur_ariman, John Daniels, Henry Thomas, Dergottdergrunten, Kristian Hiim, Sara Vieira, Zerox8610, Christopher MacDonald, Brendan, and Patrick James.

At $15 USD were Alexandre Roose, Gregory Cordell Nouch, Erik Shuttlesworth, Paul Hart, Daniel feigin, Vx Spctr, Ian Griffith, Christopher Purvis, Tanner Donovan, Scott Sells, Vincent Loibl, Sarah Linkous, Tyganit, Marcelo Gonzalez, Kimberly Crawley, Tanner Simerman, Pablo Bugarín Cámara, Marcin Nalepa, Cody Pagunsan, Jean Petrovic, Syed Zeeshan Ali, Robert Buff, An Original Clone, Benjamin Louie, Eboni Lewis, Amani Ageeli, J Churcher, Casey Adams, Todd A Zircher, Justin Loudermilk, Rashid J Buresly, Donald Pack, Chances, Patrick LaBine, Deozaan, Paul Barrett, Brad Genz, Max, Gonzalo Delgado, Austin Long, Mateusz Dworak, Thijs Moens, Gonçalo Amaral, Mohammad Iqbal, Rubén Nicolás Fanego, Filip Basara, Denis Cormier, Joeri Dehouwer, Abdullah Alsuliman, Malte Büchmann, Martin German, Renad Zakri, Damian Gańko, Jose A., Guzmán Monné, Antonio Anon Brasolin, Jamie Crompton, Dany Francis, Ricardo Delgado Torres, Lukas Isaksson, Jeferson Leite Borges, Björn Wilke, Shubham Kamble, Cody Gratner, Spencer Egart, Tweet This Dude, Jean-Dominique Baril, Terran, Marc Copes, Serhii Hrechka, Ercan Yapalak, Dgreen1220, Brenin M Melyn, Thomas Jang, Thomas Digiacomi, Cora Dale, Rodrigo Spilimbergo, Adam Doty, Alonso Montenegro, Jeremie Pinoteau, Michael Schoderer, Andrew Sit, ChrisM, Sergio Ordóñez Garrido, Wenbo Wu, Guy Kimpton, Rafal Kras-

sowski, Valerie Roussel, Matt Edmonston, Vitor Lolli, Giulia Boi, Yusak Wijaya Santoso, David Schut, Peien Wang, Jacob Wawszczyk, Alex Proctor, Mateus Plez Ricciardi, Jeremiah Goerdt, Dean Hart, Neïl Messelmani, Jeffrey Berube, Daniel Tokerød, Paul Perez, Dirk Albrecht, Dominic Scherer, Luis Teixeira, Cody Gratner, Jesse Lieberg, Jason Ormes, Olatandstad, Joshua Godwin, John Foster, Keith Atherton, Julen Irazoki, Noe Caldairou, Vitor Freitas, Larlyn T, A Green Cow, Theodoros Bourmalis, Eric Schwarzkopf, Chi, Agata Dębiec, Vojtěch Lacina, Alicia Guardeño Albertos, Patrick Zoch Alves, Ndubisi Ahukanna, Narcoleptic Newt, David Deckert, Leanne Rath, Mark Ffrench, Kévin Roussel, Paul Portanier, Marc Palkowitsch-Amberger, Flávio Grassi, Tyler Porten, Joe Frizzell, Sebastian Barrientos, Florian.hanke, Rasmus Sørensen, Pascal "Kai" Lecurieux, Tommi Partanen, Arthur Longbottom, Giannhs Tataridis, IndieGoGo, Chuck Esterbrook, Richard Handy, Sarah Kerr, Robin Finch, Todd A Zircher, Oliver Wolfond, Anthony Ancel, Théo Le Moal, Iván Ruiz Lozano, Darian Reck, Sambeau Prak, Mario Roberti, and JP V.

Together IndieGoGo backers raised a total of $24,034 USD to bring this book to life. I can't possibly thank them enough for their contributions, patience, and support.