GAME JAMMING IN UNITY

TIPS AND TRICKS TO WORK FASTER

CHRISTER KAITILA

WITH amazon appstore

CONTENTS

This book will empower you to make games fast, faster than you ever thought possible. It will teach you ultra-fast game jam workflows and uncover secret shortcuts to the finish line. Whether you're a first-timer or a veteran gamedev, this book is for you.

A LIGHT-HEARTED WARNING!

We are about to set off on a perilous adventure. We will take creative risks. We will break a lot of rules. Your peers will be shocked at your blatant disregard for the meticulous computer science you learned in school. That's okay. Rapid prototyping is not the place to seek perfection.

We will focus on speed, first and foremost, using game jamming techniques in Unity to create games in days or weeks instead of months or years. If you are an experienced software developer, some of the advice in this book may prove counterintuitive; it is meant to support experimentation while helping you get to a playable build sooner.

Just remember the sage advice of the ancient master: first learn the rules, then break the rules. Embrace the creative spirit of jazz musicians, modern artists, and beat poets: improvise, disregard your inner critic, accept imperfection, and be open to a whole lot of luck. Throw caution to the wind and dive in.

ABOUT THE AUTHOR

As a freelance gamedev since 1993, I've shipped dozens of games. I've always found that the projects took longer to complete than I wanted, so in order to increase profits, and because it was a thrill, I started to optimize my workflow speed.

After a while, I got quite fast, and one day decided to challenge myself to make 24 games in 24 months. I pulled it off and had a lot of fun. When word got out and others wanted to join the fun, I created the "One Game a Month" challenge (**onegameamonth.com**), now in its fifth year with 15,000 members.

Since then, I have slowed down a bit. I wrote a couple gamedev books, had the honor of being a keynote speaker at two Ludum Dares (**ldjam.com**), and am currently a mentor at Gamkedo (**gamkedo.com**), a game development club.

WHY RAPID PROTOTYPING?

Game jams have become a popular way for game developers to try out ideas over a few days or weeks. They are fun, often themed game development events designed to stretch your creativity by imposing an extreme time limit. They force you to cut corners, keep things simple, and avoid over-ambition. Aim too high and you'll never finish, but set your sights on a lower target and you'll find it easier to hit. Instead of designing perfectly object-oriented, methodically tested code, the wisest course of action during a game jam is to plow forward with abandon and take huge risks.

Game jams call for rapid prototyping, often resulting in a throw-away proof-of-concept for a game mechanic or interesting idea. When the jam is over—if the game concept works—it's often best to start fresh, re-implementing everything from scratch using a more professional, disciplined approach that allows you to plan and document your process.

1.1 HIT THE GROUND RUNNING WITH UNITY

Unity is perfectly suited to this kind of rapid game development. It is a game engine that comes with tools, example code, and art assets to jump-start a working prototype, helping beginners and experts alike to skip months of low-level programming. It frees you to run wild and follow your creative inspiration, knowing that anything is possible!

Unity allows you to try ideas quickly and inexpensively, then discard these experiments without regret. It supports the notion common in software development (and start-up culture) that it is better to "fail fast." By developing in a way that allows you to make risky choices, you avoid the sunk cost fallacy that often leads developers down a blind path toward failure. You are more likely to switch gears and abandon ideas that don't work if they are quick and easy to implement. If it took months to add a complex feature, you may be tempted to keep it in the game even though the feature is a failure.

THINK SMALL, NOT VERTICAL SLICES

1.2

Because a game jam imposes a time constraint, it is not the appropriate place to create a masterwork of great complexity or size. Open-world games, giant adventures, or any game that requires a large amount of content should be avoided. If the selling feature of your design is "quantity" (Huge world! Dozens of spells! Tons of quests!), you know you are headed down the wrong path. Small is good.

Similarly, don't get caught up creating a "pitch-ready" vertical slice—that is, a highly polished version of the game designed to demonstrate (usually to potential investors or publishers) what a single level in a larger game might look like. This is not the goal of most game jams, though they are often a starting point for a longer period of iterative rapid prototyping, perhaps leading to a vertical slice demo.

Never forget that a proof-of-concept should be kept simple. You're using the equivalent of duct tape to engineer something quickly and experiment with ideas; only if you keep it simple can you avoid creating an incredibly messy project, digging a hole that you may never be able to climb out of.

> *"Never forget that a proof-of-concept should be kept simple."*

HOW TO WORK **FASTER**

As we set out on our gamedev adventure, preparation and planning are key. During the iterative phase of game development, when your mock-up is still very malleable and subject to radical change, look for techniques that will help speed up your development time and allow you to try out new ideas with the least amount of effort.

TAKE THE TIME TO PLAN

LEVERAGE EXISTING RESOURCES

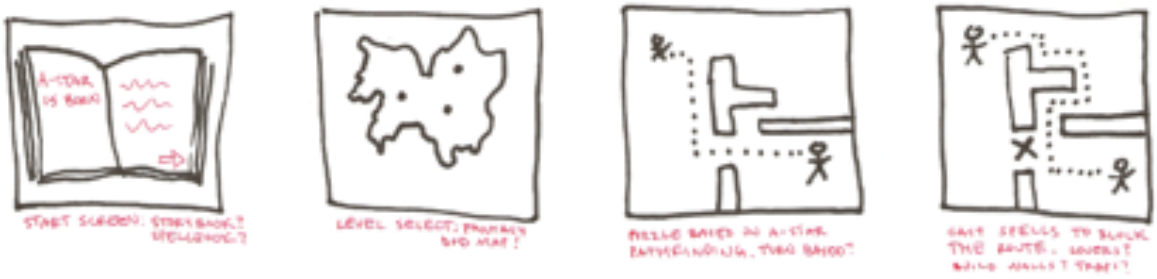
GET THE MOST FROM UNITY

When you first fire up Unity and create a new project, you immediately face that most familiar of scenes—the lovely blue sky gradient. Where do you take it from there? The possibilities are endless, of course, and this can sometimes be a real problem.

2.1 START ON PAPER

For this reason, one of the best first steps when beginning a game jam or rapid prototype is to avoid sitting in front of your computer. Instead, grab a piece of paper and a pencil, and draw a comic book version of your game.

Think of it like a storyboard. You only need stick figures, no artistic ability required.



Scribble small thumbnails for each scene when first running the game. Perhaps it begins with a splash screen, followed by a main menu. Draw them without regard for art quality but instead from a mile-high view, like making thumbnails, until you have arrived at what feels like a fun introduction to a game.

SKIP SQUARE ONE WITH "BASECODE" 2.2

What kind of camera is in use? What kind of movement will the player have? What does the world look like, and what will the moment-to-moment actions of the players be? Once you have these questions answered, you can forge ahead with the creation of your first prototype.

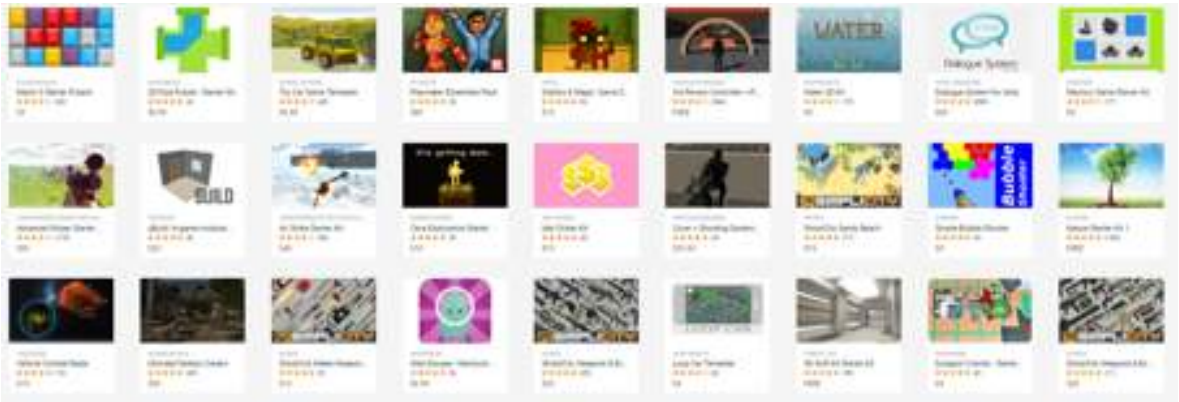
It is often handy to have your own personal basecode project to start from, which is more than just the empty default new Unity project template, but also has a few of your favorite goodies (camera scripts, your favorite save game system, or a handy scene all set up to be a main menu).

A template project, which is often referred to in game jam circles as your "basecode," or personal game template project, is an asset well worth developing in advance before proceeding with any new game project you create. After participating in several game jams, you'll begin to notice that many of the same things need solving no matter what game genre you are developing. If you don't have to start from scratch, don't!

LEVERAGE ART ASSETS 2.3

Save precious minutes or hours while developing your game by leveraging asset store goodies. An infinite variety of high-quality code and art assets are available for purchase on the Unity Asset Store and at sites such as OpenGameArt, FreeMusicArchive, and so many more.

But beware: asset stores can be both a curse and a blessing. They provide developers with a near-infinite variety of placeholder art and code solutions, but they also bring a host of negatives that need to be considered.



By using images made by many different people with varying skill levels, the art won't always mesh together. Avoid creating asset soup, a game cobbled together loosely with nothing but content created by others. Without a significant amount of your own content, you game will be merely a collage of clipart.

Instead, use asset store artwork for a mockup, game jam, proof-of-concept, or freeware fun project. It frees you to focus on the feel, and your screenshots will look awesome during development.

2.4 MASTER THE UNITY INSPECTOR AND EDIT WHILE PLAYING

The Unity inspector is where all the components in a selected object are displayed; each will have input fields such as position, rotation, and much more. Normally on the right side of the user interface, this is where you will spend countless hours changing values to get things just right.

This allows iterative design and development, since you can make edits while the game is running. In the Unity editor, you can be playing the game and still selecting objects from the hierarchy in order to view in the **Inspector** window.

Right-clicking the gear menu on the top right of any component, you can copy it to the clipboard, for use in pasting into new objects as you would expect. Additionally, a very handy alternative is to use **Paste Component Values** instead of the component itself. This will fill in the fields with values copied from another object (or the current object while it was in play mode).

Note that changes you make while the game is running take effect immediately, but aren't permanent: once you stop the game test run, you'll find that the original values return. A quick and easy workaround to this "feature" is to right-click the gear icon that opens a menu in the **Inspector** for any given component in a game object. Select the menu item labeled **Copy component** while the game is still running once you are happy with the changes.

Once you stop play-testing the game and return to the standard Unity editor mode, open up this menu again and select **Paste component values**. This will ensure that the changes you made while the game was running are saved.

This allows you to try out dozens of different values, like the position of a light or the direction that the sun is facing. Once you find values where things look perfect, you are ready to set them in stone.

UNITY QUICK TIP: INSPECTOR DEBUG MODE

The **Inspector** can reveal the private variables of your game objects. Click the gear menu in the **Inspector** and turn on **Debug** mode. This allows you to poke around in the internals of your code.

2.5 FIND STUFF FAST

As your project grows in complexity, it will become increasingly difficult to find the asset you want to edit. At this point, a few quick tips will save you tons of searching.

In the **Project Explorer** window (and also the scene hierarchy), there is a search bar. Instead of simply typing the name of something, you can refine results by the type of object. Include a type string in the search field, alongside any other words you are looking for (or none, to list all objects of that type). For example, "**t:scene**" shows only scenes, and "**t:texture**" will return only textures that match your search.

Another very handy method for quickly searching is to choose a consistent naming convention. Naming conventions are a personal choice, and open to debate, so consider this tip and see if it fits your personal style:

Prefix everything, whether textures, meshes, sounds, or code, by their general type first and specific type second, so that similar types of things end up together in alphabetical lists and dropdown menus while typing. For example, if you used "**appleGreen, appleRed,**" your searches and code completions will show all apples side-by-side in the GUI or file explorers as you start to type the word. In a file explorer window, you could start typing "**a..p..**" and see all the apples right there.

CUSTOMIZE THE UNITY EDITOR

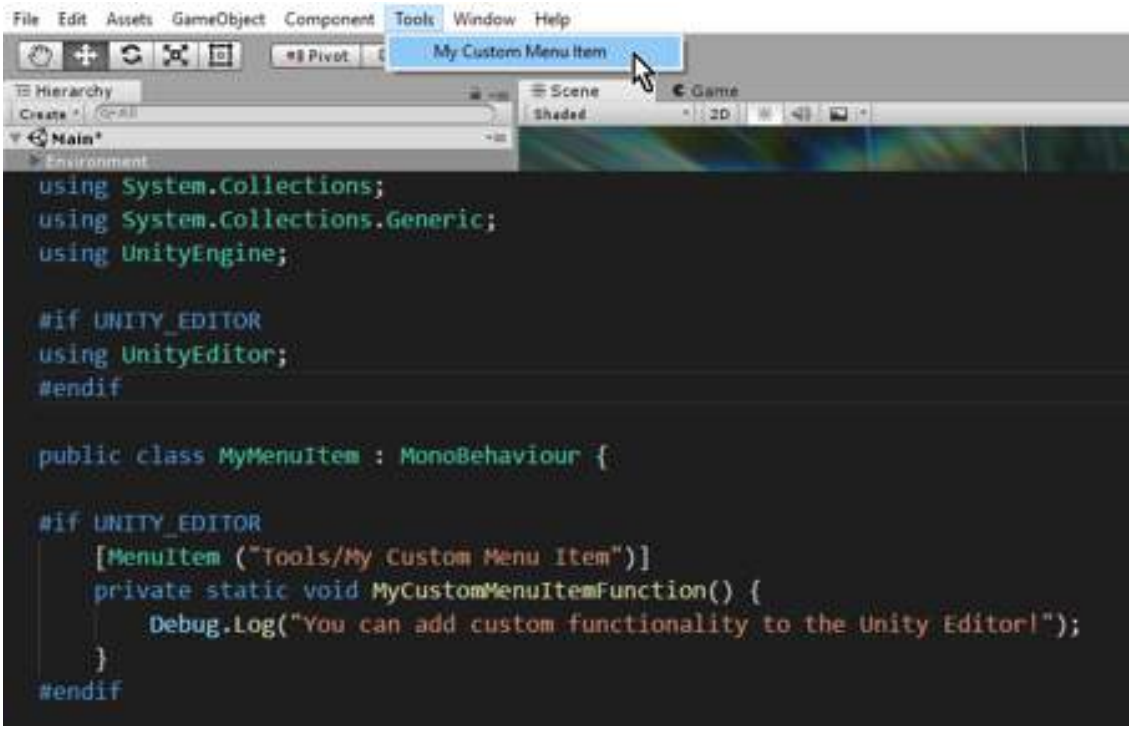
2.6

Unity empowers developers to completely redesign the entire program to suit their needs; code can be set to run in-editor, as opposed to code that runs during the game. This is the perfect way to run custom "randomization" scripts, do tests, calculate statistics, or mass-rename groups of objects while editing. Think of all the ways you can save time during the design phase!

If you place a C# source file in any folder named **Editor**, it will not be part of the game build at all and can be used for developer-only functionality. For code in other folders, you need only wrap it in a define: "**#if UNITY_EDITOR ... #endif**" to ensure it is never run during normal gameplay. This is super handy to add debug information to the game, or for custom editor functionality.

Imagine making your own tools for editing only, such as a Perlin noise terrain generator or something that rotates all selected objects to a particular orientation. To create your own menu items in the Unity editor, put the following meta tag immediately above a function: "**[MenuItem ("Tools/My Custom Function")]**".

Now, your editor enhancement can access, for example, "**Selection.gameObjects,**" an array of whichever game objects are currently selected, ready to modify as required. From there you could change "**Selection.gameObjects[0].transform.position**" or any other value.



Coding your own custom edit-mode tools is very fulfilling. You will reap great rewards, not merely in terms of time saved, but as a way to extend Unity to do anything you wish it could do.

"Coding your own custom edit-mode tools is very fulfilling. You will reap great rewards, not merely in terms of time saved, but as a way to extend Unity to do anything you wish it could do."

HOW TO CODE FASTER

The next phase is to make a playable build. A deliverable, that, no matter how simple, functions without crashing. There's a title screen, some gameplay, and a way to exit.

Reaching this stage can take some time, but the sooner you can have this up and running (even without any real gameplay or levels), the sooner you are able to start production.

RESOLVE SERIOUS BUGS IMMEDIATELY TO PRESERVE A PLAYABLE BUILD

ASSIGN TAGS AND LAYERS SO YOU ARE NOT OVERWHELMED WITH DETAIL

DEBUG AND MAKE CHANGES WHILE YOUR GAME IS RUNNING

Consider a playable build, no matter how simple, as a save point. It is a milestone along the way toward the finish line, and one that, if you were forced to abandon the game tomorrow, you would be able to return to—and present to the world as some sort of finished product. A playable build is an asset worth keeping around, backing up, and retaining as a safety net for yourself.

3.1 BUILD TOWARD MILESTONES

The key to having a playable build ready at all times is to resolve serious bugs immediately upon discovery. Don't let bugs pile up too far or you'll never get a playable build (until you resolve them all, which may only be at the very end of the project). Naturally, this is an ideal that is sometimes impossible to achieve, but keeping it in mind will serve you well.

3.2 INSULATE COMPONENTS FROM EACH OTHER

One way to avoid a total mess during the development process is to insulate various subsystems in your game engine in a defensive manner. The secret is to enforce a very thin communication channel between the separate subsystems.

Whether using events, callbacks, or some other mechanism, encapsulating objects is a safe approach to avoid complex inter-object dependencies, and is very helpful when you aren't sure how many times you'll be rewiring everything.

For example, you might separate the management of character controllers from game events and program flow. At first, you may have all game logic running within the character controller of your Player 1 game object. As you might imagine, after a while, this monolithic construct will become dependent on dozens of different scripts. Make a change in one, and suddenly five other subsystems can break.

Instead, try to ensure that different components added to your game objects require only the bare minimum of references to other components in order to function. This way you'll be free to reorder, drag-and-drop, or even delete components from your game objects without worrying that others will suddenly be left in the lurch.

UNITY QUICK TIP: AUTOSAVE

When entering play mode, Unity doesn't auto-save your work. Here's a little snippet that will ensure you don't lose anything because you forgot to save.

http://bit.ly/unityautosave

3.3 USE INSPECTOR FOR DEBUGGING AND TWEAKING

When you code a custom component for a game object, any public properties are visible and tweakable from the Unity editor's **Inspector** window.

You can take advantage of this by including a public string variable that contains debug information, a value that you can change easily over the course of the game. This serves as a quick-and-dirty real-time update when you don't want to fill the debug console log with thousands of lines of text.

For example, you might create a public string to hold the AI state of a bot, allowing you to watch it during play in the inspector, rather than implementing a new widget or custom inspector in a whole new script file.

3.4 AUTOMATICALLY RUN CODE WHILE EDITING LEVELS

Complex prefabs may have all sorts of things that need to be updated based on events triggered in the code when properties change. You don't want to have to re-run the game every time you make a change to see the result.

The easy solution for when you want to run some code in reaction to changes is to listen for the **OnValidate()** event. Here you can "refresh" the object as needed.

Create a function with this name in a component on the object in question and any time you change a value in the inspector (for example, health), you can run any code required. For example, you could automatically change the color of a healthbar sprite based on the number you typed, even though the game is not running. This technique also helps avoid the need for "custom inspectors" just to implement a "recalculate stuff" button, saving much time and code.

ASSIGN TAGS AND LAYERS

3.5

As a project grows in size, it becomes more important to hide some of the details from yourself so that you're not inundated with too much on the screen while working.

By selecting groups of objects and giving them a tag or setting their layer in the **Inspector** window, you'll be able to quickly view or select objects of only a certain type in game, even while it is running.

Ensuring that your game objects in every scene are assigned a tag or layer can also improve the efficiency of collision detection and click event recasting (if the Collision Flags are set properly to ignore objects you know will never need to be checked).

UNITY QUICK TIP: DESTROY SOMETHING LATER

Instead of coding complex timestamp, event, or coroutine logic to wait to remove an object until after a sound or animation gets played, you can delay the object's inevitable destruction by including the number of seconds to wait as the second parameter. For example, **Object.Destroy(objectToDestroy, delayInSeconds)**. This quick shortcut saves you time and extra code when a simple delay is enough.

HOW TO MAKE ART FASTER

Now we are going to explore ways to create and iterate on the content assets of your game. This includes level designs, textures, shaders, sounds, music, particle systems, and anything else that makes up the look and feel of your game.

CREATE MOCKUPS BEFORE COMMITTING TO POLISHED ASSETS

—

EFFICIENTLY WORK WITH PREFABS

—

QUICKLY NAVIGATE THE UNITY EDITOR ENVIRONMENT

—

The best way to start building your world is to make a greybox, or "block mesh" of a level, a simplified representation of the eventual shapes the level will take. Once the locations of large objects and the flow of the level's movement "feels right," then detailed artwork can be swapped in.

4.1 SET THE SCENE WITH GREYBOXING

Before texturing or lighting, working on the rough greybox of level design allows you to move large objects around, test the movement flow, and tweak distances between things. Make big layout changes during the greybox stage and you avoid the hassle, encouraging quick iteration because it's still easy to do before the world has been made to look beautiful.

UNITY QUICK TIP: SNAPPING OBJECTS

When you need your prefabs to line up perfectly, you will want to snap them to a grid. Pressing **[Ctrl]** while moving an object snaps the position to full world units (coordinates with no decimals, just multiples of one). **[Ctrl-Shift]** will snap the object to any collision that results from a raycast from the cursor into the scene from the editor camera angle. This will help you place trees and rocks on the terrain. This is one of the most handy level design workflow tricks.

BREAK COMPLEX SCENES INTO STAGING AREAS WITH PREFABS

4.2

Prefabs are everything. They empower you to define a **GameObject** as the template for a bullet or enemy, for example, and then spawn hundreds of copies around your game levels. When you edit the prefab, every copy also changes.

Some developers only use prefabs for objects that need to be duplicated en masse, but it can be very handy to make almost everything a prefab, even objects for which only one copy ever exists in-game. (Static level meshes might be the exception.)

Why? Because you can work on them in smaller “staging” scenes. This comes in handy when tuning performance. It also helps when editing especially complex levels where it is too easy to click nearby objects. This allows you to focus on one thing, with perfect editor framerate and fewer mis-clicks.

Breaking complex scenes into staging areas also simplifies team collaboration, since GitHub merge conflicts are very common in large Unity scenes. The technique outlined above avoids this by allowing people to work in separate files more often. Imagine, for example, a racing game level. An artist can work on the layout of the track scene while another dev works on the physics for the car in a temporary staging scene. The level makes reference to the car prefab, so the new changes propagate during development—with fewer merge conflicts, since neither person was editing the same file on disk.

UNITY QUICK TIP: MULTI-EDIT

You can select multiple objects and any attributes they share (such as the transform), to be edited all at once. For example, say you have ten thousand statues in your level but they were all sitting at different altitudes. Instead of moving each one manually, you can select them all and simply type "**0**" into the **Y** input box. Every object will move as required! Other fields (**X** and **Z**, for example) will remain untouched and individual to each object in the selection.

This group "mass editing" functionality is not well known, but once you know it, you'll use it often. It is handy even for simple things like making many objects static, or giving them all the same material—all without having to make the same minor change to hundreds of objects, one at a time.

AVOID UNITY DEFAULTS

4.3

Few things make a game look cheaper than a splash screen with no image or a game executed with the standard Unity icon. This shows a lack of polish. The same holds true for other frequent mistakes such as using the default GUI styles, or the default Unity sky (the most familiar blue gradients). Disappointingly, many game jam games do not even have the game name filled out in the title bar of the game, and if you view the executable's file properties, you can see there is no company name nor other details listed. With this in mind, remember to replace all of the standard empty defaults associated with new Unity projects. Open the build settings and fill in values for the name of your game, company (or your internet handle), your game's icon, and resolution selection splash screen header image (if you use that GUI at startup).

UNITY QUICK TIP: COME HERE, OBJECT!

Moving objects in Unity can be a drag, literally. It can be hard to figure out which way to move something depending on the camera angle. One very easy way to put things exactly where you want, facing the exact angle you wish, is to fly the camera around until it sits where you want the new object to be. With the object selected, use the menu command **GameObject...Align With View** and it will move to the exact location and rotation of the camera.

This is also a very handy "come here" command. Use it to move far-off objects near to where you are currently working, then you can place them more easily. You won't waste time looking around for them in space, then returning to the location you are concerned with.

THE FINISH LINE

Near every finish line is the dreaded wall. That point where the most challenges await you, the point when lesser game developers give up and abandon their games. In this chapter, we outline mechanisms and development methodologies that help you climb over the wall. Shortcuts, safety nets, and a healthy attitude will carry you past that final troublesome bugfix.

UNDERSTAND YOUR GAME MAY NEVER BE PERFECT

PRIORITIZE MUST-HAVE FEATURES

IDENTIFY WHICH FEATURES CAN BE CUT

Declaring your game "done" and ready for the public requires swallowing your pride. Rejecting the goal of perfectionism and humbly accepting your limitations. It is not easy.

5.1 KNOW WHEN YOUR GAME IS DONE

Even if your next game is a huge hit and even if the press clamor to interview you and shower you with accolades, it is quite likely that you'll be secretly obsessing over all the things in your game you wish you'd polished just a little bit more.

Perfection is both a valuable guiding star in the night and the nightmare in your sleep. It presents the direction you need to travel toward. It also leads to your doom. To succeed as a gamedev, you need to fail as a perfectionist.

"Perfection is both a valuable guiding star in the night and the nightmare in your sleep."

PRIORITIZE MUST-HAVES AND REJECT NICE-TO-HAVES

5.2

When you're working on something that inspires you, you'll find it easy to come up with dozens (or even hundreds) of cool ideas you wish you had time to implement. The practical reality of finite time and resources will force you to discard almost all of your ideas, and this can be a painful process. It's hard to cast aside concepts simply because you don't have time to implement them all.

One great way to manage your risk and ensure that you'll hit the finish line is to force yourself to rank all ideas for new features or additional content. List them as must-haves (mission-critical features that the game simply could not function without) and nice-to-haves (wish-list items that might add a little extra something to the game, but must be chosen with moderation).

The vast majority of wonderful ideas that you put into your nice-to-have wish list will never be implemented, and that's a good thing! In the same way that perfection is the enemy of complete, complexity, quantity, and detail are walls in the path toward your finish line.

5.3

DETERMINE WHICH FEATURES CAN BE CUT

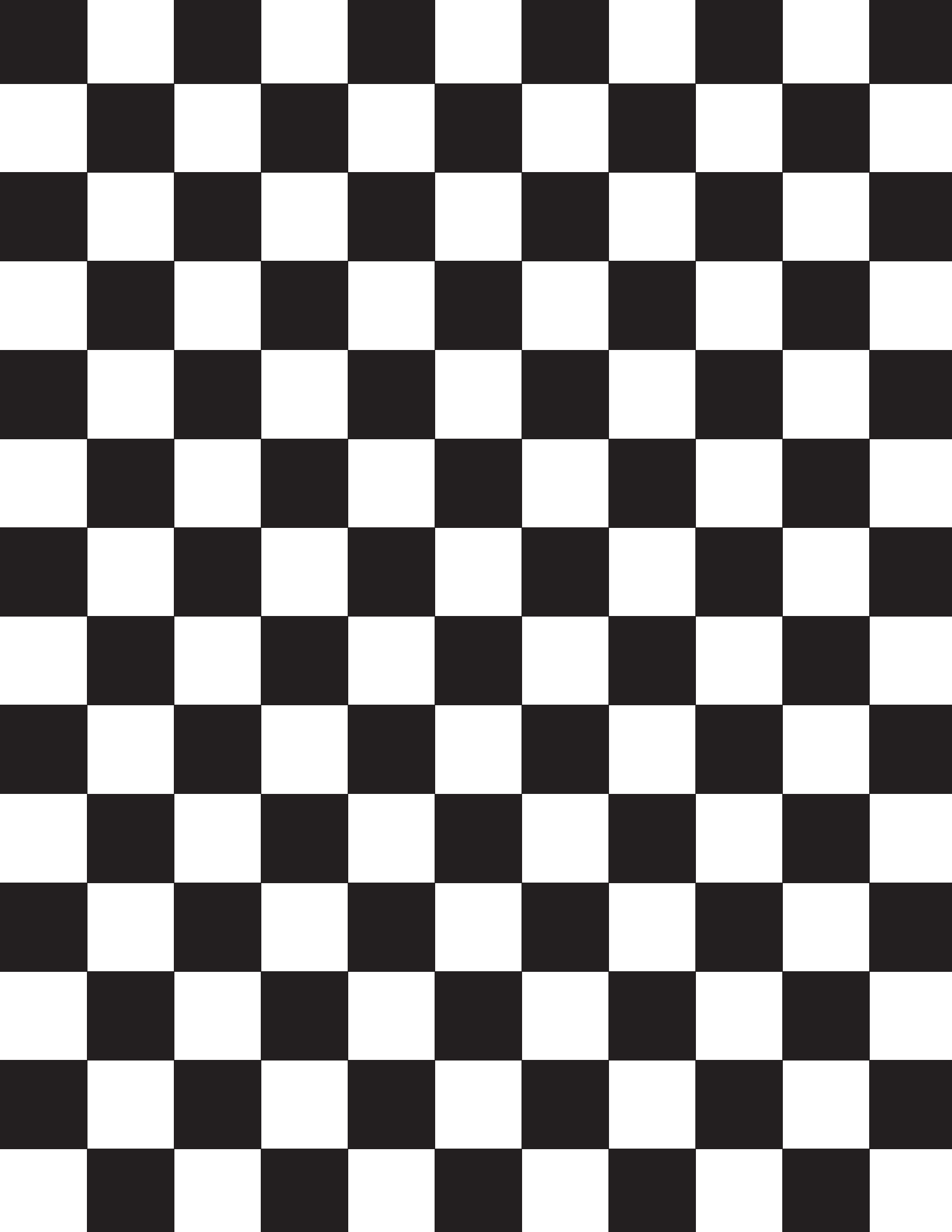
When a game is nearing completion, features are always cut. Why? The practicalities of getting the game out the door—fewer features, less work to do.

To determine which features can be cut, consider the following for each in your game:

- **Does the feature improve the core loop of gameplay?**
- **Is the feature important to the story/theme of the game?**
- **Will polishing the feature to perfection be easy or work intensive?**
- **Do other games do this better or is it unique and exciting?**
- **Is completing this feature a side quest or a main quest?**

When in doubt, cut that feature. Short on time? Cut another feature. Running out of money? Cut yet another feature. Does your shooter really need an inventory and crafting system? Does your RPG really need seventeen-character classes?

When you're not sure whether to add something new or iterate on something already in the game, choose to iterate. Don't add. Subtract.



CONCLUSION

Remember that a mockup, MVP, prototype, or vertical slice is not the same thing as a finished game. Like a prototype motor in an engineering lab covered in duct tape, a rapidly prototyped game is often rather unstable. There will be missing parts. There will be tech debt. There will be sections in the code needing optimization and polish.

Reap the rewards of rapid iteration; you found ways to "fail fast" that were cheap and easy. Now that you know what works, you can rebuild slowly and with a higher level of quality. A game jam game, like a sandcastle, is ephemeral. Quick prototypes are never meant to last. They prove a concept, show a promising game mechanic, or are the takeaways from an exciting weekend. If you want to take your game to the next level, you'll need to move beyond the rapid prototype stage.

Like an artist's sketch, you now have the opportunity to tear down the prototype and refactor your game—this time around, built solidly, without as much experimentation.

NOTE FROM THE AUTHOR

As your expertise grows, your workflow will improve. You're better at what you do, but less flexible. You know more solutions, but have fewer fresh ideas.

For this reason, I warmly encourage you to remember "what it felt like" when you were just starting out; embrace the idea that anything is possible. Some call this "beginner's mind." It is one of the secrets to my success, and I hope this wide-eyed, open-mindedness pervades the book.