Reddit for Game Makers

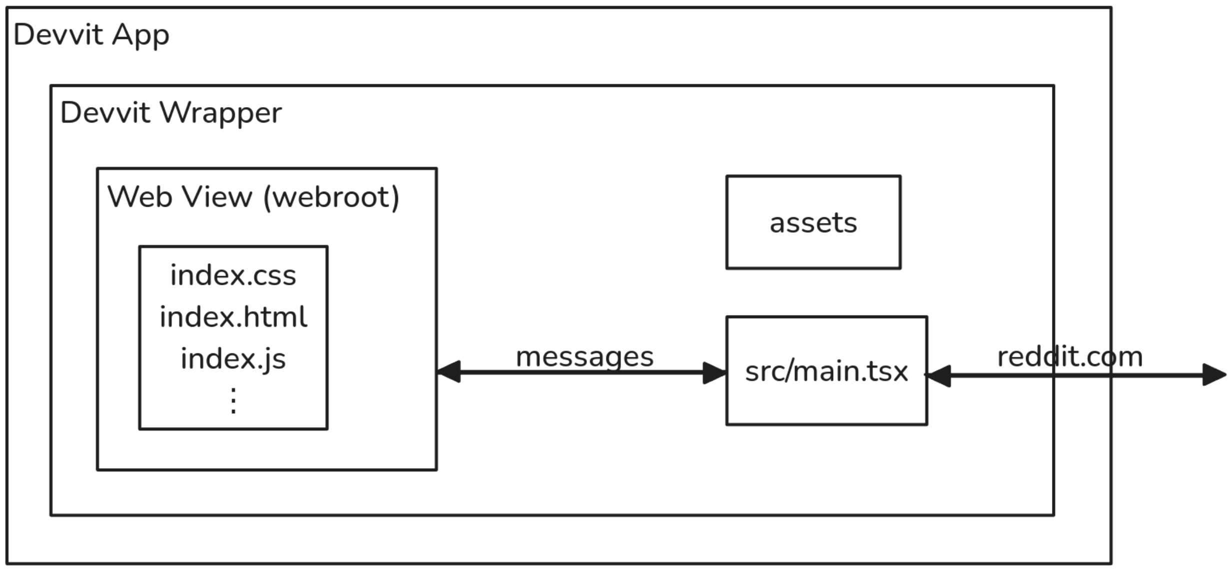
A handbook for Devvit, the new platform for building and running games on reddit.com.

The purpose of this document is to:

- Encourage every developer to use **live-reload on reddit.com** and **local development**.
- Clarify big games can have **small Devvit integrations**.
- Catalog miscellaneous tips and pitfalls.

App Architecture

Devvit apps are NPM projects with a thin JavaScript wrapper around a conventional **<iframe>** called a web view.



A typical project looks like the following.

- **assets/**: Devvit wrapper assets.
- **src/**
  - **main.tsx**: (required) Devvit wrapper entry point.
- **webroot/**: All web view code and assets.
  - **index.html**: Web view document HTML.
  - **index.js**: Web view script referenced by index.html.
- **devvit.yaml**: (required) App name and version.
- **package.json**: (required) NPM dependencies and scripts.

TypeScript / JavaScript is only required for the wrapper. webroot/ can contain JavaScript or any code compiled to Wasm from languages like C++, GDScript, and Lua. Most games can be written to run entirely in the web view with only a light Devvit wrapper. Code under webroot/ has no platform integration except iframe messages.

> 💡 Tips:
>
> - Maximize webroot/ code. It's extremely portable and easily tested without reddit.com. For example, game engines like Godot support Wasm export. The sources for a Godot game may exist within or without the project. Only the Wasm needs to be output to webroot.
> - **devvit-hub** is an unofficial library that abstracts a lot of the boilerplate wanted by most games.
> - The Devvit wrapper UI framework is called Blocks. Blocks code may be prototyped quickly in **play** if plugin data is faked.

Files under webroot/ must be ready for use by the browser. For example, these cannot be TypeScript files because TypeScript must be compiled to JavaScript first. Unlike webroot, src/main.tsx and any imports are compiled by the Devvit CLI.

See **TypeScript and JavaScript Apps** for details on structuring a game written in TypeScript / JavaScript.

Development Workflows

Both live-reload and local development flows are essential to working fast.

Live-reload on reddit.com

The Devvit CLI supports automatic live-reload and local client / remote server console log aggregation (*except* logs emitted from the web view) via **`devvit playtest`**. You can enable both by appending the `?playtest` query parameter to your browser URL `https://reddit.com/r/<subreddit>?playtest`.

A typical workflow is:

1. Execute `npx devvit playtest r/<subreddit>`.
2. Playtest initializes.
3. Open `https://reddit.com/r/<subreddit>?playtest` in your local web browser.
4. Playtest reports a connection to the browser is made.
5. Edit and save any file under src/, webroot/, or assets/.
6. Playtest uploads and installs the new version and streams logs from both server and client execution.
7. Playtests reloads the browser automatically and reconnects.

8. Interact with the app and open the browser DevTools logs.

Each iteration is “hands free” and, for small uploads, should cost about 30 seconds.

⚠️ Warnings:

- Complete the **quickstart**.
    - Create a dedicated private subreddit for playtesting. This is both to avoid distributing broken versions and to limit execution logs to users expected.
- Also create a dedicated development version of the app as devvit.dev.yaml and playtest with the `--config=devvit.dev.yaml` argument. Web views assets are unversioned so this is the only way to separate production and development versions.
- Logs emitted from the web view (any code under webroot/) only appear in the browser console, not the CLI.
- Playtesting requires at least one app post to be loaded.

💡 Tips:

- The `--playtest=<app>` query parameter can filter to a single app if there are multiple apps installed in the test subreddit.
- Playtesting usually works best on a specific post page to avoid getting logs from other posts. Eg, `https://reddit.com/r/chippeddev/comments/1haenvy/rock_c7d05f7_at_beryozovskoye_deposit/?playtest`.
- Playtest with `--log-runtime` to label local client and remote server logs.

Extend Live-reload to Include Custom Build Processes

Intuitively, a change to any source file should trigger a new build-upload-install-reload loop but the Devvit CLI doesn’t know of any custom build processes needed to generate files under webroot/. Any such build processes should be wired into the playtest workflow. For example, a React web view app may have TypeScript sources under src/web-view that must be compiled to JavaScript outputs in webroot/.

This example uses esbuild to compile TypeScripts sources under src/web-view/index.ts to webroot/index.js. Since a playtest session is started in parallel, any changes emitted by esbuild to webroot/ will trigger a new app version upload.

```
Unset
$ npx devvit playtest r/<subreddit> &
$ playtest=$!
$ npx esbuild \
  --bundle \
  --format=esm \
  --log-level=info \
  --outfile=webroot/index.js \
  --watch=forever \
  src/web-view/index.ts &
$ esbuld=$!
$ trap 'kill $playtest $esbuild' sigint
$ wait
```

See also this **NPM script example**.

Local Development

Playtesting on reddit.com requires a lengthy build-upload-install-reload loop so develop locally whenever possible. The following setup is needed:

- A local HTTP server that can serve files from webroot/.
- A build process that can either output to webroot/ or serve from memory.
- A mechanism to live-reload the browser or hot-replace code changed.
- Fake Devvit messages.

This example uses esbuild to compile TypeScript sources to JavaScript in memory, serve image, audio, and other assets under webroot/, and live-reload the browser.

```
Unset
$ npx esbuild \
  --banner:js='new EventSource("/esbuild").addEventListener("change", () =>
location.reload());' \
  --bundle \
  --format=esm \
  --log-level=info \
  --outfile=webroot/index.js \
  --serve=1234 \
  --servedir=webroot \
  --watch=forever \
  src/web-view/index.ts
```

Each iteration costs about 2 seconds and decouples development from as many Devvit-isms as possible.

See also this **NPM script example**.

Fake Devvit Messages

Fake messages from the Devvit Blocks integration are used to prime and control the web view.

```javascript
const app = new App() // The web view app or game.
app.start() // Register message, frame, and other listeners.

// Development mode detection varies on development server. Eg,
// `process.env.NODE_ENV === 'development`.
const noDevvit = location.port === '1234'
if (noDevvit) {
  // Fake an init message received at some random time.
  setTimeout(
    () =>
      engine._onMsg(
        new MessageEvent<DevvitSystemMessage>('message', {
          data: {
            type: 'devvit-message',
            data: {
              message: {
                save: {
                  score: Math.random() * 100,
                  snoovatarURL:

'https://i.redd.it/snoovatar/avatars/a67a8a09-fb44-4041-8073-22e89210961d.png',
                  t2: 't2_k6ldbjh3',
                  username: 'stephenoid'
                }
              }
            }
          }
        })
      ),
    Math.random() * 1_000
  )
}
```

💡 Tips:

- Copy an example message by logging it from a prior reddit.com session.
- **Typecheck messages** (`MessageEvent<DevvitSystemMessage>` above) to keep reddit.com and local development behavior identical as enforced by the compiler.
- Vary message injection time with Math.random() to simulate real world variability in loading.

Debugging the Web View

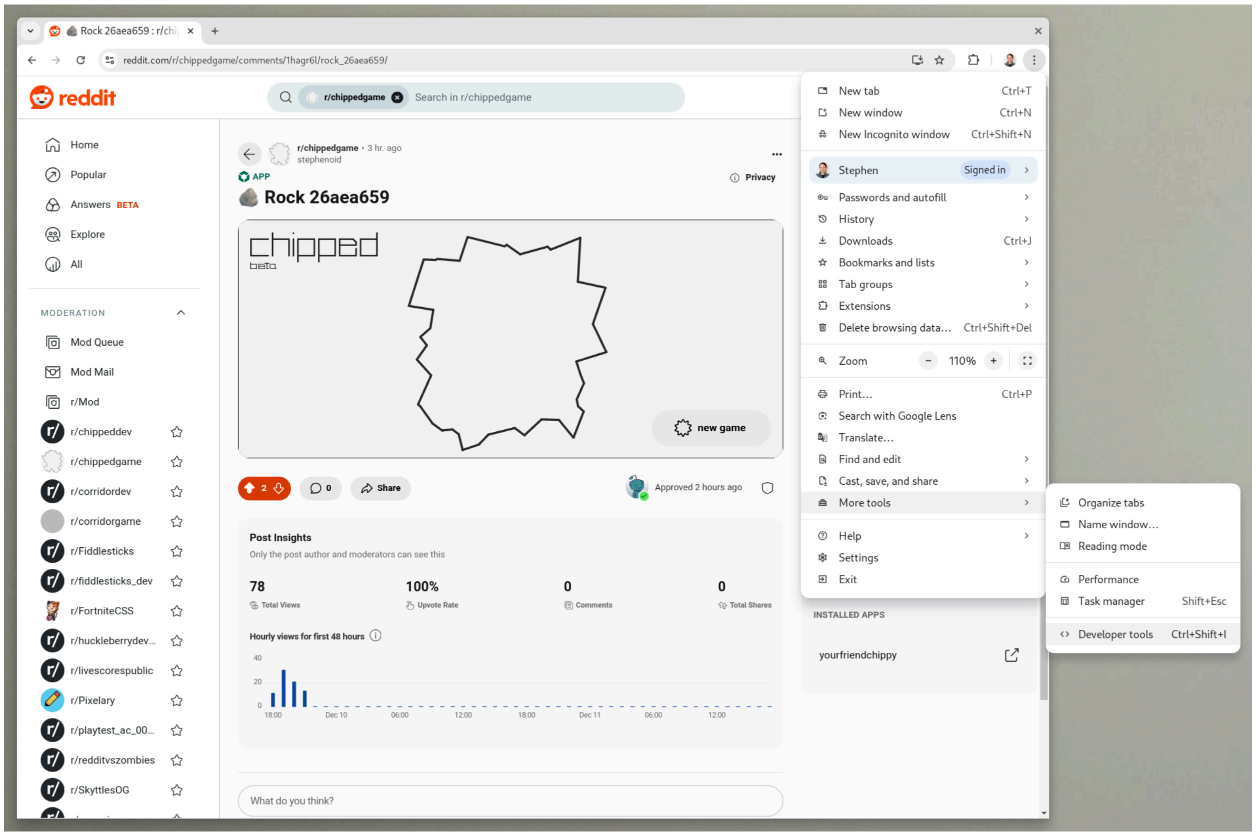
Logs emitted from the web view (anything under webroot/) appear only in the browser console.

Output a Message to the Browser Console

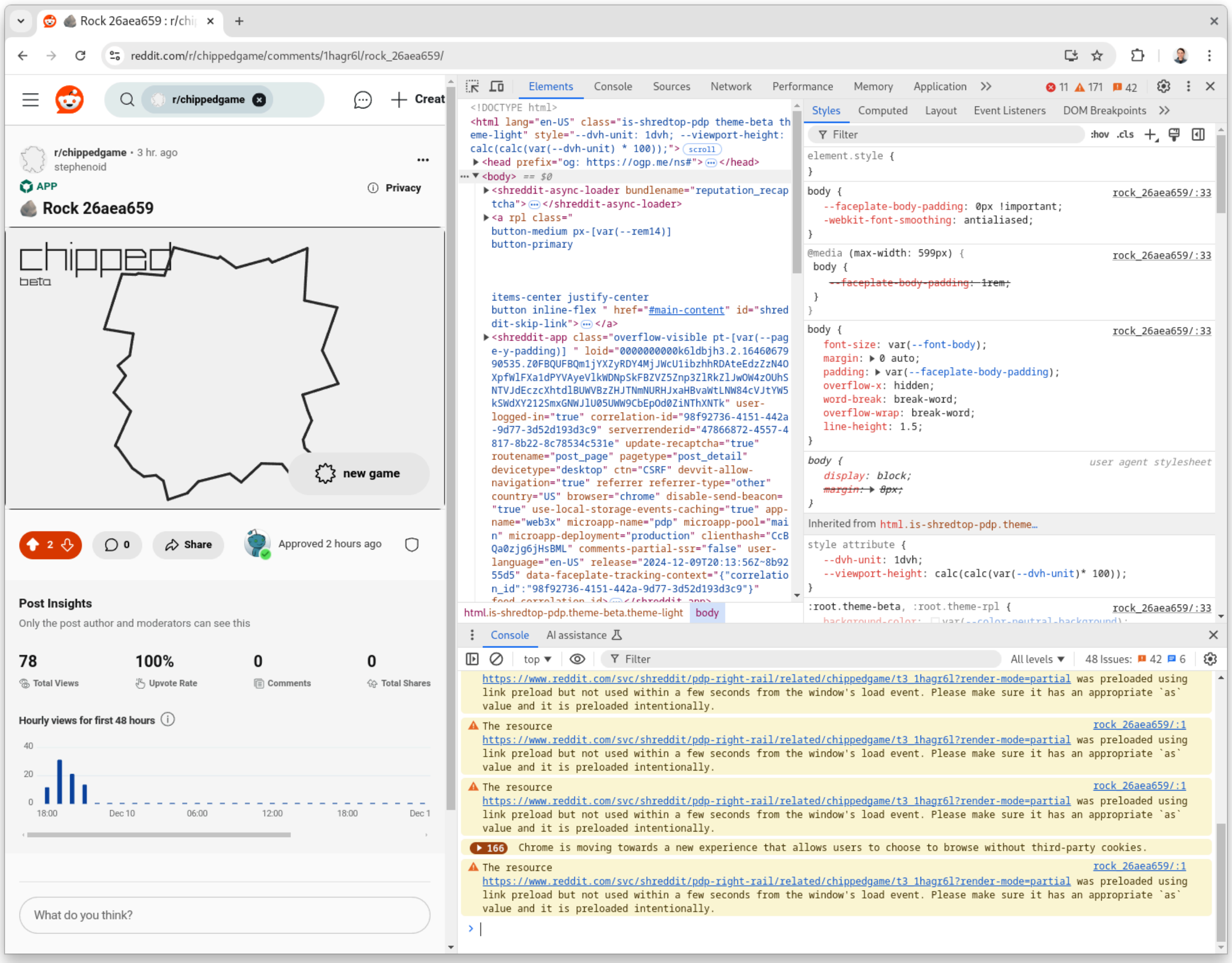
```javascript
console.log('hello')
```

Connect the Chrome Debugger (DevTools)

1. Press F12 or ctrl-shift-i or press overflow -> More tools -> Developer tools.



2. Press escape to open the DevTools browser console.



```
<!DOCTYPE html>
<html lang="en-US" class="is-shredtop-pdp theme-beta theme-light" style="--dvh-unit: 1dvh; --viewport-height: calc(calc(var(--dvh-unit) * 100));"> scroll
 ▶<head prefix="og: https://ogp.me/ns#"> ··· </head>
··· ▼<body> == $0
   ▶<shreddit-async-loader bundlename="reputation_recaptcha"> ··· </shreddit-async-loader>
   ▶<a rpl class="
    button-medium px-[var(--rem14)]
    button-primary


    items-center justify-center
    button inline-flex " href="#main-content" id="shreddit-skip-link"> ··· </a>
   ▶<shreddit-app class="overflow-visible pt-[var(--page-y-padding)] " loid="0000000000k6ldbjh3.2.1646067990535.Z0FBQUFBQm1jYXZyRDY4MjJWcU1ibzhhRDAteEdzZzN4OXpfWlFXa1dPYVAyeVlkWDNpSkFBZVZ5Znp3ZlRkZlJwOW4zOUhSNTVJdEczcXhtdlBUWVBzZHJTNmNURHJxaHBvaWtLNW84cVJtYW5kSWdXY212SmxGNWJlU05UWW9CbEpOd0ZiNThXNTk" user-logged-in="true" correlation-id="98f92736-4151-442a-9d77-3d52d193d3c9" serverrenderid="47866872-4557-4817-8b22-8c78534c531e" update-recaptcha="true" routename="post_page" pagetype="post_detail" devicetype="desktop" ctn="CSRF" devvit-allow-navigation="true" referrer referrer-type="other" country="US" browser="chrome" disable-send-beacon="true" use-local-storage-events-caching="true" app-name="web3x" microapp-name="pdp" microapp-pool="main" microapp-deployment="production" clienthash="CcBQa0zjg6jHsBML" comments-partial-ssr="false" user-language="en-US" release="2024-12-09T20:13:56Z~8b9255d5" data-faceplate-tracking-context="{"correlation_id":"98f92736-4151-442a-9d77-3d52d193d3c9"}"
html.is-shredtop-pdp.theme-beta.theme-light   body
```

```
element.style {
}
body {                                                    rock_26aea659/:33
   --faceplate-body-padding: 0px !important;
   -webkit-font-smoothing: antialiased;
}
@media (max-width: 599px) {                              rock_26aea659/:33
 body {
   ~~--faceplate-body-padding: 1rem;~~
 }
}
body {                                                    rock_26aea659/:33
   font-size: var(--font-body);
   margin: ▶ 0 auto;
   padding: ▶ var(--faceplate-body-padding);
   overflow-x: hidden;
   word-break: break-word;
   overflow-wrap: break-word;
   line-height: 1.5;
}
body {                                           user agent stylesheet
   display: block;
   ~~margin: ▶ 8px;~~
}
Inherited from html.is-shredtop-pdp.theme…
style attribute {
   --dvh-unit: 1dvh;
   --viewport-height: calc(calc(var(--dvh-unit)* 100));
}
:root.theme-beta, :root.theme-rpl {                       rock_26aea659/:33
```

```
https://www.reddit.com/svc/shreddit/pdp-right-rail/related/chippedgame/t3_1hagr6l?render-mode=partial was preloaded using
link preload but not used within a few seconds from the window's load event. Please make sure it has an appropriate `as`
value and it is preloaded intentionally.
⚠ The resource                                                                                      rock_26aea659/:1
https://www.reddit.com/svc/shreddit/pdp-right-rail/related/chippedgame/t3_1hagr6l?render-mode=partial was preloaded using
link preload but not used within a few seconds from the window's load event. Please make sure it has an appropriate `as`
value and it is preloaded intentionally.
⚠ The resource                                                                                      rock_26aea659/:1
https://www.reddit.com/svc/shreddit/pdp-right-rail/related/chippedgame/t3_1hagr6l?render-mode=partial was preloaded using
link preload but not used within a few seconds from the window's load event. Please make sure it has an appropriate `as`
value and it is preloaded intentionally.
▶ 166  Chrome is moving towards a new experience that allows users to choose to browse without third-party cookies.
⚠ The resource                                                                                      rock_26aea659/:1
https://www.reddit.com/svc/shreddit/pdp-right-rail/related/chippedgame/t3_1hagr6l?render-mode=partial was preloaded using
link preload but not used within a few seconds from the window's load event. Please make sure it has an appropriate `as`
value and it is preloaded intentionally.
> |
```

Add a Debug Pane to the Document

JavaScript

```javascript
/**
 * @type {readonly unknown[]} args
 * @return {void}
 */
function logHTML(...args) {
  let pre = document.querySelector('#dbg')
  if (!pre) {
    pre = document.createElement('pre')
    pre.id = 'dbg'
    pre.style.position = 'absolute'
    pre.style.bottom = '0'
    pre.style.width = '100%'
    pre.style.height = '100px'
    pre.style.overflow = 'auto'
    document.body.append(pre)
  }
  pre.innerText = `${args.join(' ')}\n${pre.innerText}`
}
logHTML('a', 'b', 'c')
```

Vibrate a Message

JavaScript

```
navigator.vibrate?.(100)
navigator.vibrate?.([
  100, 30, 100, 30, 100, 30, 200, 30, 200, 30, 200, 30, 100, 30, 100, 30, 100,
])
```

⚠️ Warning: vibrate() is unavailable on iOS.

Beep a Message

JavaScript

```
/**
 * @arg {readonly number[]} durations
 * @return {void}
 */
function beep(...durations) {
  const ctx = new AudioContext()
  let t = ctx.currentTime
  for (const millis of durations) {
    const osc = ctx.createOscillator()
    osc.connect(ctx.destination)
    osc.start(t)
    osc.stop(t + millis / 1000)
    t += .1 + millis / 1000
  }
}
beep(100, 200, 100)
```

Relay a Message to `devvit` Logs

Relaying messages from the web view to the Devvit wrapper logs to the Devvit CLI instead of the browser console. This is useful when the debugger cannot be connected or seeing all logs in one place is wanted.

```javascript
// Web view.
/**
 * @arg {readonly unknown[]} args
 * @return {void}
 */
function dlog(...args) {
  parent.postMessage({type: ‘Log’, args: args.map(arg => JSON.serialize(arg,
undefined, 2))}, ‘*’)
}
dlog('a', 'b', 'c')

// Devvit wrapper.
<webview
  id='web-view'
  onMessage={msg => {
    switch (msg.type) {
      case 'Log':
        // Relay web view message to the CLI logs.
        console.log(...msg.args)
      break
    }
  }}
  url='index.html'
/>
```

⚠️ Warning: relaying messages is slow and error prone.

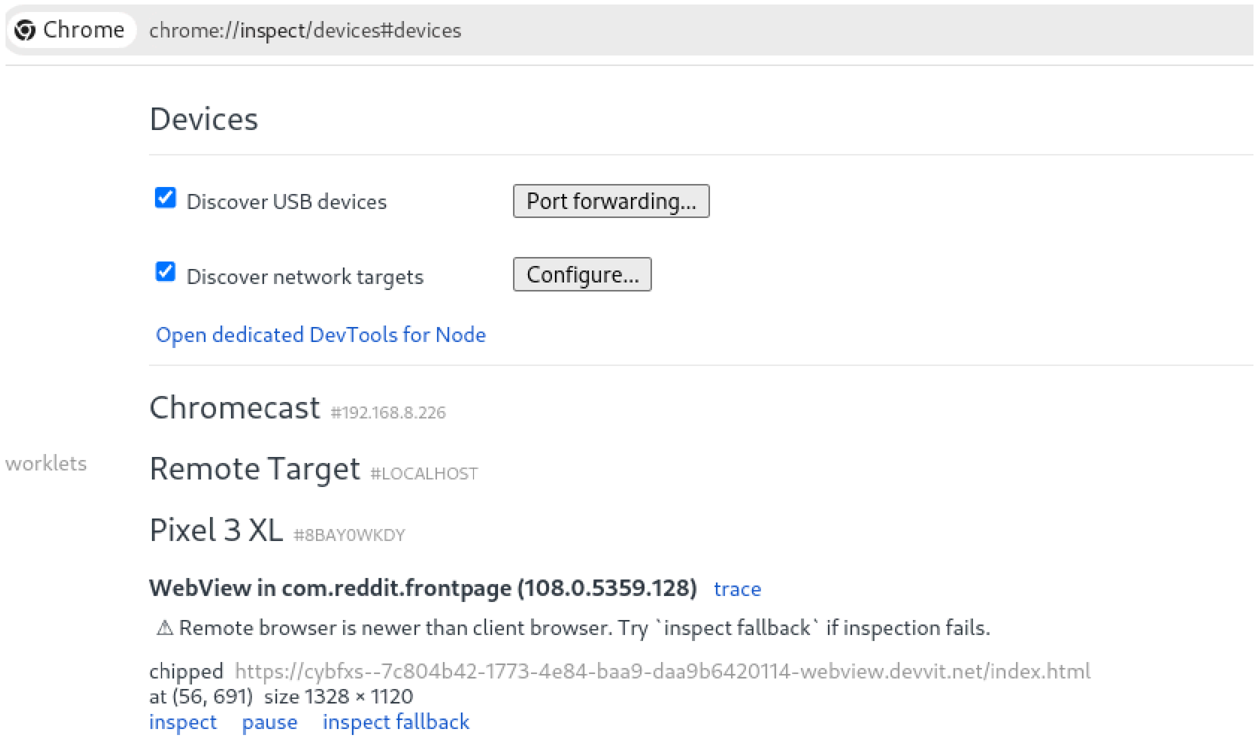
Pause the Debugger

When a debugger is attached such as Chrome DevTools, the `debugger` statement will pause execution allowing the program’s running state to be inspected. The statement has no effect when a debugger is not attached.

```javascript
console.log('hello')
if (Math.random() > .5) debugger
console.log('goodbye')
```

Connect the Android Debugger with Chrome (DevTools)

1. Launch the native Android Reddit app or Chrome for Android.
2. Open an app post and launch the web view.
3. Launch Chrome on desktop.
4. Visit `chrome://inspect`.
5. Look for the correct session under com.reddit.frontpage (Reddit for Android) or Chrome for Android.



Chrome chrome://inspect/devices#devices

Devices

☑ Discover USB devices [Port forwarding...]

☑ Discover network targets [Configure...]

Open dedicated DevTools for Node

worklets

Chromecast #192.168.8.226

Remote Target #LOCALHOST

Pixel 3 XL #8BAY0WKDY

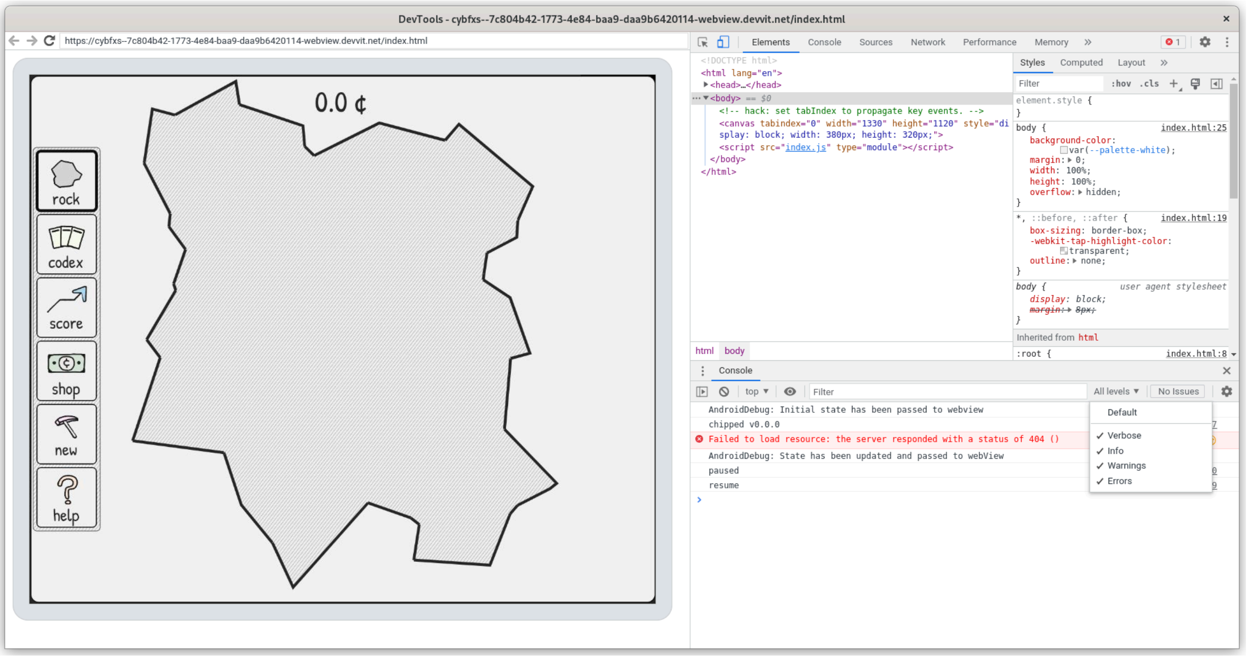
WebView in com.reddit.frontpage (108.0.5359.128) trace

⚠ Remote browser is newer than client browser. Try `inspect fallback` if inspection fails.

chipped https://cybfxs--7c804b42-1773-4e84-baa9-daa9b6420114-webview.devvit.net/index.html
at (56, 691) size 1328 × 1120
inspect pause inspect fallback

6. Click inspect.

7. Press escape to open the browser DevTools console.

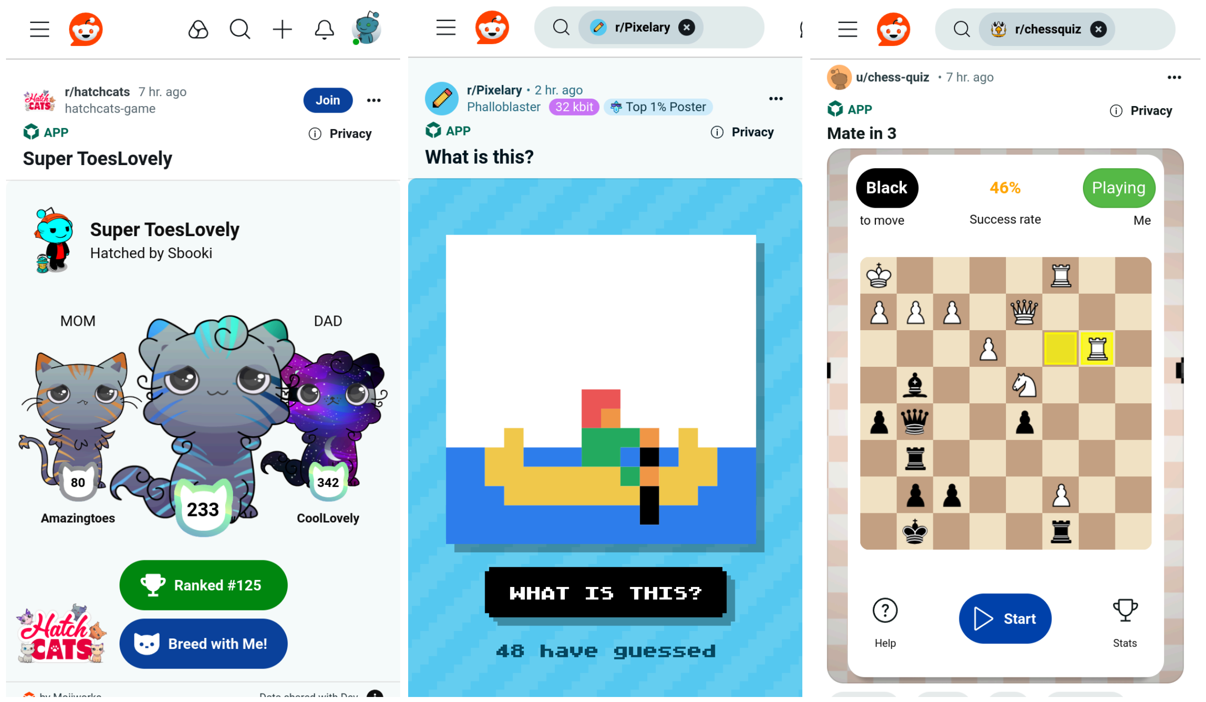

DevTools - cybfxs--7c804b42-1773-4e84-baa9-daa9b6420114-webview.devvit.net/index.html
https://cybfxs--7c804b42-1773-4e84-baa9-daa9b6420114-webview.devvit.net/index.html
0.0 ¢
rock
codex
score
shop
new
help
Elements
Console
Sources
Network
Performance
Memory
Styles
Computed
Layout

```
<!DOCTYPE html>
<html lang="en">
▸<head>…</head>
▾<body> == $0
    <!-- hack: set tabIndex to propagate key events. -->
    <canvas tabindex="0" width="1330" height="1120" style="display: block; width: 380px; height: 320px;">
    <script src="index.js" type="module"></script>
  </body>
</html>
```

```
element.style {
}
body {                                          index.html:25
    background-color:
        var(--palette-white);
    margin: ▸ 0;
    width: 100%;
    height: 100%;
    overflow: ▸ hidden;
}
*, ::before, ::after {                          index.html:19
    box-sizing: border-box;
    -webkit-tap-highlight-color:
        transparent;
    outline: ▸ none;
}
body {                              user agent stylesheet
    display: block;
    ~~margin: ▸ 8px;~~
}
Inherited from html
:root {                                          index.html:8
```

```
AndroidDebug: Initial state has been passed to webview
chipped v0.0.0
Failed to load resource: the server responded with a status of 404 ()
AndroidDebug: State has been updated and passed to webView
paused
resume
>
```

Design



Hatch Cats, Pixelary, and ChessQuiz are apps with strong differentiating screens per post, a simple and inviting interaction, games across many posts, and social gameplay.

Surfaces

All Devvit games are exposed through posts and comments. The paradigm is unusual compared to retro console games and is an opportunity to create something not yet seen.

Social

Reddit is a community of communities. We think most games on Reddit should be sociable:

- Leaderboard.
  A single-player game with a leaderboard is the easiest social feature to implement.
- Multiplayer.
  See **implementing multiplayer**.
- Posts and comments.
  These fundamental mediums of Reddit can be inputs and outputs of the game.

Distinctive First Screens with a Clear Call to Action

Posts with a unique starting screen clarify how each new play is different and stand out better in a feed. It should be clear and easy to start playing.

> 💡 Tip: consider recording a unique random number seed with each post to generate the same experience for all users visiting that post (**example**).

Playable Across Posts Within a Community

Reddit doesn't have an arcade and users are unlikely to find a single ancient post. Apps that either encourage user content generation or generate their own posts are more shareable and fit better into the existing post ecosystem. (Content generation can be scheduled using **the scheduler API**.) Consolidating posts in a dedicated community helps users find them.

Responsive UI, Dark and Light Mode, and Pointer Input

Reddit is everywhere. Apps should be built to look great on desktop, mobile web, native Android, and native iOS phones, tablets, and computers of all sizes in dark and light mode and be navigable with at least pointer input. Mobile devices account for most of Reddit's traffic.

Snoovatars

Players have a strong Reddit identity and many have highly customized snoovatars. Consider using them in game for avatars and leaderboards (**example**).

Customizing the Loading Screen

When an app creates a post from the Devvit wrapper, it can set the preview property to customize the load screen.

JavaScript
```
const post = await ctx.reddit.submitPost({
  preview: <Preview foo='bar' />,
  subredditName: ctx.subredditName,
  title: `unique post title ${Math.random()}`
})
```

The property is a component rendered at post submission time.

JavaScript
```
export function Preview(props: {foo: string}): JSX.Element {
  return (
    <vstack
      width='100%'
      height='100%'
      alignment='center middle'
      backgroundColor='#123'
    >
      {props.foo}
      <image
        url='loading.gif'
        description='loading…'
        height='140px'
        width='140px'
        imageHeight='240px'
        imageWidth='240px'
      />
    </vstack>
  )
}
```

⚠️ Warnings:

- App loading screens are part of the post state, not app version. Changes made to the loading screen will not be seen in old posts unless **Post.setCustomPostPreview()** is called on each. This is true when playtesting a specific post as well; you must create a new post or update an existing one to see any changes made.
- Apps do not have access to Context nor plugins. If data is needed, it must be passed by property at time of post submission.

💡 Tips:

- Consider using a data URI to generate a distinctive custom loading screen that transitions seamlessly from loading to loaded (**example**).
- The title screen of an app may share a component with the loading screen to make the transition to loaded seamless. Pass any data needed in the shared component by property.
- Use **play** to prototype different loading screens quickly.

Implementing Multiplayer

Devvit's **realtime API** provides peer-to-peer messaging over sockets which can be used for player interactions of all kinds.

💡 Tips:

- Improve the *perception* of responsiveness using prediction, reconciliation, and interpolation. There are many articles on techniques to improve multiplayer experiences such as **Fast-Paced Multiplayer: Client-Server Game Architecture** (**demo**). Think critically about how to handle client contention such as two players reaching for the same loot.
- Consider synchronizing audio to UTC if supporting multiple devices in the same room.
- Inject fake peer messages to simulate multiplayer locally (**example**).

⚠️ Warnings:

- Apps can receive messages from peers on older (and newer) versions of the app that haven't refreshed recently. It's hard to code against older and future message types so consider filtering messages processed to only those matching the current version of the app. You can also detect if a user is running an older version and prompt them to reload (**example**).
- Any app messages an app sends is received by all instances of the app including the sender instance. Filter out any unexpected messages to yourself by recording the sender UUID in each message (**example**).
- Players may open multiple sessions of an app simultaneously. Use a UUID to distinguish each session instead of a Reddit user ID (T2).
- App messages are broadcasted all posts. If per-post messaging is wanted, add the post ID to the channel name (**example**).
- Optimize message payloads and minimize message frequency. The realtime API does not tolerate overuse.
- Clients can execute on peer messages received but only server executed Devvit wrapper code can send messages.
- Peers may be disconnected unexpectedly without notice. Use a heartbeat and timeout to remove stale peers (**example**).
- Apps should avoid making plugin calls in response to peer messages. For example, fetching the peer's snoovatar. Instead, apps should either cache the data in Redis or send it in the message from the peer (**example**). This avoids multiplying the work needed by the app (and reddit.com) by the number of peers which may result in ratelimiting.
- Unsubscribe when an app is backgrounded to avoid being ratelimited (**example**). It's also best practice to disable rendering to reduce power and computational resource

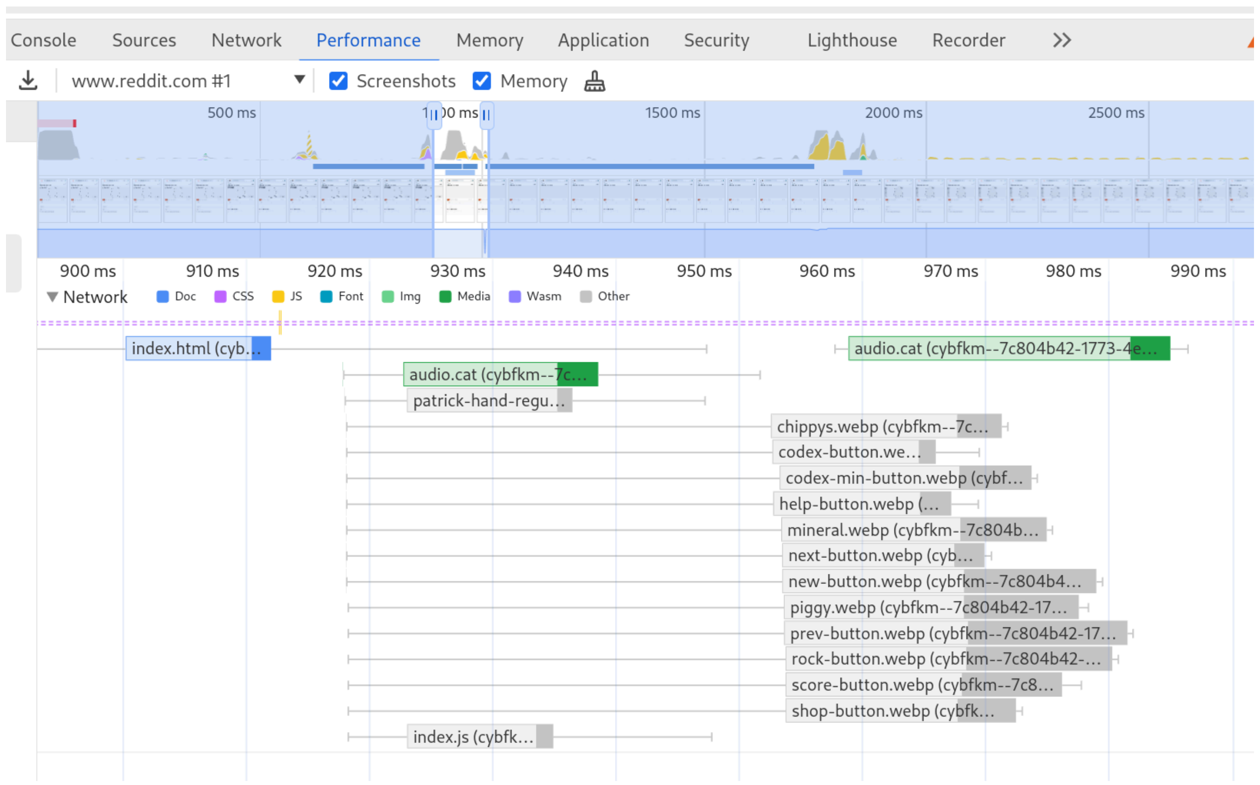
usages (**example**).

Optimizing Page Load

Preload known assets in parallel to your JavaScript or wasm wasm bundle instead of waiting to `fetch` required assets. Request them at HTML parse time using **preload links**.

Unset
```
<html>
  <head>
    <link rel='preload' href='assets/audio-atlas.mp3' as='fetch' />
    <link rel='preload' href='assets/font.ttf' as='font' type='font/woff2'
crossorigin />
    <link rel='preload' href='assets/image-atlas.webp' as='image'
type='image/webp' />
  </head>
  <body>
    <script src='index.js' type='module' async></script>
  </body>
</html>
```

We can download index.js and several other assets at the same time. When index.js is downloaded and executed, the preloaded assets will be ready or already downloading.



In the Chrome DevTools' performance tab, everything is blocked by the initial index.html load. Once available, audio.cat, patrick-hand-regular.ttf, index.js, and numerous images are able to initiate downloads at the same time instead of being blocked on index.js. If preloads were not used, index.js would have to finish loading and execute first before additional assets would be requested.

See this [larger preload example](#).

💡 Tips:

- See [Customizing the Loading Screen](#) to improve the perception of performance.

TypeScript and JavaScript Apps

Games can be made entirely in TypeScript or JavaScript. This works well for TypeScript / JavaScript game engines and developers with prior web experience.

TypeScript Configuration

Most TypeScript apps probably want at least five distinct TypeScript configurations:

- **./**tsconfig.json
  The root tsconfig; contains references to all the other configs which allows typechecking the all sub-projects.
- **src/devvit/**tsconfig.json
  The Devvit wrapper; an ES2020 web worker-like environment with no DOM access.

- **src/shared**/tsconfig.json
  Code shared between the web view and Devvit wrapper, such as messages, in a common denominator ES2020 web worker-like environment.
- **src/test**/tsconfig.json
  A Node.js v22.x environment for tests and utils in all folders.
- **src/web-view**/tsconfig.json
  Web view: the full ES2020 DOM environment of an iframe.

This immense boilerplate is necessary to describe the different environments accurately and enable project-wide typechecking (**example**).

Typecheck All Messages Between the Devvit Wrapper and the Web View

Typecheck messages to keep reddit.com and local development behavior identical as enforced by the compiler. Typechecking is critical to keeping local development fast and accurate. The following is an example of a message contract shared between the Devvit wrapper and the web view.

```JavaScript
// src/shared/messages.ts

type DevvitSystemMessage = {
  readonly type?: 'devvit-message'
  data: {message: DevvitMessage}
}

/** A message from the Devvit wrapper to the web view. */
type DevvitMessage = {
  /** Player one save state. */
  save: Save
  readonly type: 'Init'
} | {
  /** A remote player has joined or updated their state. */
  player: Player
  readonly type: 'PlayerUpdate'
}

/** A message from the web view to the Devvit wrapper. */
export type WebViewMessage =
  | {readonly type: 'Loaded'}
  | {readonly type: 'Save'; p1: Player}
```

With these definitions, validate typing in the web view.

```
JavaScript
// src/web-view/index.ts

function onMsg(ev: MessageEvent<DevvitSystemMessage>): void {
  // Filter unknown messages.
  if (ev.data.type !== 'devvit-message') return

  const msg = ev.data.data.message
  switch (msg.type) {
    case 'Init':
      // to-do: initialize game state.
      break
    case 'PlayerUpdate':
      // to-do: update player state.
      break
    default:
      // Type-check that all messages are handled.
      msg.type satisfies never
  }
}
addEventListener('message', onMsg)

// Only expect supported messages so everything else fails type-checking.
function postMessage(msg: WebViewMessage): void {
  parent.postMessage(msg, '*')
}
postMessage({type: 'Loaded'})
```

And in the Devvit wrapper.

```
JavaScript
// src/main.tsx

// Only expect supported messages.
function onMsg(msg: WebViewMessage): void {
  switch (msg.type) {
    case 'Loaded':
      // Web view is listening. initialize it. use the type parameter to verify
      // typing.
      ctx.ui.webView.postMessage<DevvitMessage>('web-view', {type: 'Init',
save: {score: 123}})
    break
    case 'Save':
      // to-do: save data to Redis.
    break
    default:
      // Type-check that all messages are handled.
      msg.type satisfies never
  }
}

// Elsewhere, create the web view with onMessage().
<webview
  id='web-view'
  onMessage={msg => onMsg(msg as WebViewMessage)}
  url='index.html'
  width='100%'
  height='100%'
/>
```

Pointer Input

The Pointer API exposes both mouse and touch state in a single interface that behaves the same on desktop and mobile devices with this single line of CSS:

Unset
```
canvas {
  /* Update on each pointermove *touch* Event like *mouse* Events. */
  touch-action: none;
}
```

Pointer capture can be used on pointer down events to limit mouse input to the canvas to prevent accidentally clicking elsewhere until the next pointer up or cancel event (**example**). The more aggressive **Pointer Lock API** can be used instead to also capture pointer down and require the user to press escape to regain control.

💡 Tip: hide the cursor with `cursor: none;` and draw a custom cursor for a more immersive experience. Be careful that the cursor hitbox and pointer up / down states are obvious.

Keyboard, Game Pad, and Joystick Inputs

Keyboards, game pads, and joysticks are supported (**example**).

Pixelated and Pixel Perfect Games

Games with a pixelated look almost certainly want to enable nearest-neighbor canvas scaling.

```
Unset
canvas {
  image-rendering: pixelated;
}
```

Pixel perfect games will likely have to consider at least the following:

- Window or client size (`innerWidth` and `innerHeight`).
- Body dimensions (`width` and `height`), `margin`, and `overflow`.
- Canvas attribute dimensions (`width` and `height`).
- Canvas `width` and `height` *style* properties.
- Browser zoom and `devicePixelRatio`.
- Minimum camera size and the camera transform.
- Integral scaling.

Tips and Pitfalls

- .ogg is unsupported on iOS.
- DXC-911 webp translucency is broken in the Devvit wrapper (but works fine in web views).
- CanvasRenderingContext2D.createPattern() does not support spritesheets.
- Web, native Android, and native iOS can be distinguished (**example**).
- Don't forget to `await` any asynchronous code returning a Promise (nearly all plugins are async).