Package 'videogameinsightsR'

October 27, 2026

Title Interface to 'Video Game Insights' API for Gaming Market Analytics

Version 0.3.0

Author Phillip Black [aut, cre]

Maintainer Phillip Black <pblack@gameeconomistconsulting.com>

Description Interface to the 'Video Game Insights' API v4 <https://app.sensortower.com/vgi/> for video game market analytics. Retrieves game metadata, developer and publisher information, player statistics (concurrent players, daily and monthly active users), revenue and units, reviews, wishlists, followers, price history, player overlap and player demographics for games on 'Steam', 'PlayStation' and 'Xbox', as tidy tibbles with VGI and Steam identifiers for cross-platform comparison.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Imports dplyr, httr2, jsonlite, digest, rlang, tibble, tidyr, rvest, stats, magrittr

Suggests dotenv, ggplot2, gt, httptest2, knitr, rmarkdown, scales, testthat, withr

Config/testthat/edition 3

URL https://github.com/econosopher/videogameinsightsR

BugReports https://github.com/econosopher/videogameinsightsR/issues

NeedsCompilation no

Repository CRAN

Date/Publication 2026-10-27 12:40:02 UTC

Contents

.vgi_cache_dir . . . 3
apply_rate_limit . . . 3
create_rate_limiter . . . 4
get_steam_app_id . . . 5
print.vgi_yoy_comparison . . . 5
should_use_rate_limiting . . . 6
vgi_active_players_by_date . . . 7
vgi_all_developer_games . . . 9
vgi_all_games_player_overlap . . . 10
vgi_all_games_playtime . . . 11
vgi_all_games_regions . . . 12
vgi_all_games_top_countries . . . 13
vgi_all_games_wishlist_countries . . . 14
vgi_all_publisher_games . . . 15
vgi_auth_check . . . 17
vgi_build_cache . . . 17
vgi_cache_stats . . . 18
vgi_clear_cache . . . 18
vgi_concurrent_players_by_date . . . 19
vgi_developers_overview . . . 21
vgi_developer_games . . . 22
vgi_developer_info . . . 23
vgi_developer_list . . . 24
vgi_fetch_all_games . . . 25
vgi_fetch_by_ids . . . 25
vgi_filter_genre . . . 26
vgi_followers_by_date . . . 27
vgi_games_metadata . . . 29
vgi_game_list . . . 30
vgi_game_metadata . . . 31
vgi_game_metadata_batch . . . 32
vgi_game_summary . . . 33
vgi_game_summary_yoy . . . 34
vgi_get_games_by_id . . . 36
vgi_historical_data . . . 37
vgi_historical_data_by_date . . . 38
vgi_insights_ccu . . . 40
vgi_insights_dau_mau . . . 41
vgi_insights_followers . . . 42
vgi_insights_player_regions . . . 43
vgi_insights_playtime . . . 44
vgi_insights_price_history . . . 44
vgi_insights_revenue . . . 45
vgi_insights_reviews . . . 46
vgi_insights_units . . . 47
vgi_insights_wishlists . . . 48

vgi_load_cache 49
vgi_peak_ccu_by_ids 49
vgi_peak_ccu_by_names 50
vgi_player_overlap 50
vgi_price_history 51
vgi_publishers_overview 52
vgi_publisher_games 53
vgi_publisher_info 54
vgi_publisher_list 55
vgi_revenue_by_date 56
vgi_reviews_by_date 58
vgi_search_games 60
vgi_smart_game_search 61
vgi_steam_market_data 63
vgi_top_countries 63
vgi_top_regions 64
vgi_top_wishlist_countries 65
vgi_units_sold_by_date 66
vgi_wishlists_by_date 68

Index **71**

`.vgi_cache_dir` *Video Game Insights Smart Cache System*

Description

Provides intelligent caching for Steam App IDs and game metadata to minimize API calls while enabling local filtering by genre, platform, and other attributes.

Usage

```
.vgi_cache_dir()
```

`apply_rate_limit` *Apply Rate Limiting to Date Sequence*

Description

Simple function to add rate limiting to date-based loops

Usage

```
apply_rate_limit(
  date_position,
  total_dates,
  calls_per_batch = NULL,
  delay_seconds = NULL,
  show_messages = TRUE
)
```

Arguments

date_position	Current position in date sequence
total_dates	Total number of dates
calls_per_batch	Calls per batch (default from environment)
delay_seconds	Delay between batches (default from environment)
show_messages	Whether to show messages

Value

No return value. Called for side effects (optional delay) and returns `invisible(NULL)`.

---

create_rate_limiter *Create a Rate Limiter*

Description

Creates a rate limiter object that tracks API calls and applies delays when thresholds are reached.

Usage

```
create_rate_limiter(
  calls_per_batch = 10,
  delay_seconds = 1,
  show_messages = TRUE
)
```

Arguments

calls_per_batch	Number of calls before applying delay
delay_seconds	Seconds to pause between batches
show_messages	Whether to show rate limiting messages

Value

A rate limiter environment with increment() method

Examples

```
limiter <- create_rate_limiter(
  calls_per_batch = 5, delay_seconds = 0, show_messages = FALSE
)
for (i in 1:10) limiter$increment()
limiter$get_stats()
```

`get_steam_app_id` *Get Steam App ID from Steam Store Search*

Description

Searches the Steam store and returns the app ID of the first result. Useful for finding Steam IDs when you only know the game name.

Usage

```
get_steam_app_id(game_name)
```

Arguments

`game_name`	Character string. The game name to search for.

Value

Integer. The Steam App ID of the first search result, or NULL if not found.

Examples

```
## Not run:
# Find Steam ID for a game
bf_id <- get_steam_app_id("Battlefield 2042 Open Beta")
print(bf_id)

## End(Not run)
```

`print.vgi_yoy_comparison` *Print Method for Year-over-Year Comparison*

Description

Print Method for Year-over-Year Comparison

Usage

```
## S3 method for class 'vgi_yoy_comparison'
print(x, ...)
```

Arguments

x	A vgi_yoy_comparison object.
...	Additional arguments passed to print().

Value

The input object x, invisibly.

---

should_use_rate_limiting

Check if Rate Limiting Should Be Applied

Description

Determines if rate limiting should be applied based on the expected number of API calls.

Usage

```
should_use_rate_limiting(num_dates, num_metrics = 1)
```

Arguments

num_dates	Number of dates to process
num_metrics	Number of metrics being fetched

Value

Logical indicating if rate limiting is recommended

vgi_active_players_by_date *Get Active Players Data by Date*

Description

Retrieve daily and monthly active user (DAU/MAU) data for all games on a specific date, providing engagement metrics across the market.

Usage

```
vgi_active_players_by_date(
  date,
  steam_app_ids = NULL,
  offset = NULL,
  limit = NULL,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

date	Character string or Date. The date for which to retrieve data in "YYYY-MM-DD" format.
steam_app_ids	Numeric vector. Optional Steam App IDs to filter results. If not provided, returns data for all available games.
offset	Integer. How many results to skip over. Minimum is 0. Optional.
limit	Integer. Maximum number of results to return. Minimum is 1, maximum is 1000. Optional.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Details

Active player metrics provide deeper engagement insights than concurrent players:

- DAU: Unique players who launched the game on this date
- MAU: Unique players who launched the game in the past 30 days
- DAU/MAU ratio: Key metric for player retention (higher = better)
- Industry benchmark: 10-20\

These metrics are essential for:

- Measuring true player engagement

- Identifying games with strong retention
- Tracking seasonal patterns
- Comparing engagement across genres
- Forecasting player trends

Value

A data frame with columns:

steamAppId Integer. The Steam App ID

date Character. The date of the data

dau Integer. Daily active users

mau Integer. Monthly active users

dauMauRatio Numeric. DAU/MAU ratio (engagement rate)

activeRank Integer. Rank by DAU

Examples

```
## Not run:
# Get active player data
active_data <- vgi_active_players_by_date("2024-01-15")

# Top 20 games by DAU
top_dau <- head(active_data, 20)
cat("Top 20 games by daily active users:\n")
print(top_dau[, c("steamAppId", "dau", "mau", "dauMauRatio")])

# Find games with excellent retention (DAU/MAU > 25%)
high_retention <- active_data[active_data$dauMauRatio > 0.25 &
                              active_data$dau > 1000, ]
cat("Games with >25% DAU/MAU ratio:", nrow(high_retention), "\n")
print(head(high_retention[order(-high_retention$dauMauRatio), ], 10))

# Compare with concurrent player data
ccu_data <- vgi_concurrent_players_by_date("2024-01-15")
engagement <- merge(active_data, ccu_data, by = "steamAppId")

# Calculate concurrent-to-DAU ratio (session intensity)
engagement$ccu_dau_ratio <- engagement$peakConcurrent / (engagement$dau + 1)

# Games with high session intensity (many concurrent per daily user)
high_intensity <- engagement[engagement$ccu_dau_ratio > 0.3 &
                             engagement$dau > 1000, ]
cat("High session intensity games:", nrow(high_intensity), "\n")

# Analyze retention tiers
active_data$retention_tier <- cut(active_data$dauMauRatio,
                                  breaks = c(0, 0.1, 0.2, 0.3, 0.5, 1),
                                  labels = c("Poor", "Below Avg", "Good",
                                             "Excellent", "Outstanding"))
```

```
retention_summary <- table(active_data$retention_tier[active_data$dau > 100])
barplot(retention_summary,
        main = "Games by Retention Tier (DAU > 100)",
        xlab = "Retention Tier",
        ylab = "Number of Games",
        col = rainbow(5))

# Monthly trend analysis
month_ago <- as.Date("2024-01-15") - 30
active_prev <- vgi_active_players_by_date(as.character(month_ago))

trend <- merge(active_data, active_prev,
               by = "steamAppId",
               suffixes = c("_now", "_prev"))

# Calculate monthly growth
trend$dau_growth <- ((trend$dau_now - trend$dau_prev) /
                     (trend$dau_prev + 1)) * 100
trend$mau_growth <- ((trend$mau_now - trend$mau_prev) /
                     (trend$mau_prev + 1)) * 100

# Find rapidly growing games
growing <- trend[trend$dau_growth > 50 & trend$dau_now > 1000, ]
cat("Games with >50% DAU growth:", nrow(growing), "\n")

# Identify games losing players
declining <- trend[trend$mau_growth < -20 & trend$mau_prev > 5000, ]
cat("Games losing >20% MAU:", nrow(declining), "\n")
print(head(declining[order(declining$mau_growth),
           c("steamAppId", "mau_prev", "mau_now", "mau_growth")]))

## End(Not run)
```

`vgi_all_developer_games` *Get Game IDs for Developers*

Description

Retrieve the VGI game IDs attached to each developer from `/companies/developers/game-ids`. Games on every platform are included; use `vgi_developer_games()` for one developer's Steam App IDs.

Usage

```
vgi_all_developer_games(
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list(),
  vgi_ids = NULL,
```

```
  slugs = NULL,
  limit = NULL,
  cursor = NULL,
  all_pages = FALSE
)
```

Arguments

auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.
vgi_ids	Integer vector. VGI company IDs to select.
slugs	Character vector. VGI company slugs to select.
limit	Integer. Records per page (maximum 1000).
cursor	Integer. Cursor from a previous page.
all_pages	Logical. Fetch every page.

Value

A tibble with columns developer_id, name, game_ids (list-column of VGI game IDs) and game_count; see vgi_all_publisher_games().

Examples

```
## Not run:
vgi_all_developer_games(vgi_ids = 28663)

## End(Not run)
```

---

vgi_all_games_player_overlap

Get Player Overlap Across the Catalogue

Description

Walk /player-overlap game by game, starting after cursor. The endpoint returns one game per request, so limit games cost limit requests and each row can be large. For one game use vgi_player_overlap().

Usage

```
vgi_all_games_player_overlap(
  limit = 1,
  cursor = NULL,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

limit	Integer. Number of games to return.
cursor	Integer. VGI ID after which to start (the next_cursor attribute of a previous call).
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

A tibble with one row per game: vgi_id, steam_app_id, top_overlaps (list-column of overlap tibbles, see vgi_player_overlap()), overlap_count, top_overlap_game (VGI ID) and top_overlap_pct, sorted by top_overlap_pct. The attribute next_cursor carries the cursor for the next game.

Examples

```
## Not run:
vgi_all_games_player_overlap(limit = 5)

## End(Not run)
```

---

vgi_all_games_playtime

Get Playtime for Many Games (v4)

Description

Retrieve playtime statistics for many games from the v4 /player-insights/games/playtime endpoint, optionally restricted to players in given countries or regions. For one Steam game use vgi_insights_playtime().

Usage

```
vgi_all_games_playtime(
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list(),
  steam_app_ids = NULL,
  vgi_ids = NULL,
  slugs = NULL,
  countries = NULL,
  regions = NULL,
  limit = NULL,
  cursor = NULL,
  all_pages = FALSE
)
```

Arguments

auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.
steam_app_ids	Integer vector. Steam App IDs to select. Optional.
vgi_ids	Integer vector. VGI internal game IDs to select. Optional.
slugs	Character vector. VGI game slugs to select. Optional.
countries	Character vector of 2-letter ISO country codes. Limits the numbers to players from those countries.
regions	Character vector of VGI region slugs. Limits the numbers to players from those regions.
limit	Integer. Games per page (API default 200, maximum 1000).
cursor	Integer. Cursor from a previous page.
all_pages	Logical. Follow the cursor through every page.

Value

A tibble with one row per game: vgi_id, platform, steam_app_id, avg_playtime, median_playtime (minutes), avg_playtime_rank, avg_playtime_prct, playtime_ranges (list-column of tibbles with range, percentage) and playtime_rank (rank by average playtime within the returned rows). The attribute next_cursor carries the cursor for the next page.

Examples

```
## Not run:
vgi_all_games_playtime(steam_app_ids = 4019220, countries = "US")

## End(Not run)
```

---

vgi_all_games_regions *Get Player Regions for Many Games (v4)*

Description

Retrieve the player-share-by-region breakdown for many games from the v4 /player-insights/games/top-regions endpoint, widened to one column per region. For one Steam game use vgi_top_regions().

Usage

```
vgi_all_games_regions(
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list(),
  steam_app_ids = NULL,
  vgi_ids = NULL,
```

```
  slugs = NULL,
  limit = NULL,
  cursor = NULL,
  all_pages = FALSE
)
```

Arguments

auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.
steam_app_ids	Integer vector. Steam App IDs to select. Optional.
vgi_ids	Integer vector. VGI internal game IDs to select. Optional.
slugs	Character vector. VGI game slugs to select. Optional.
limit	Integer. Games per page (API default 200, maximum 1000).
cursor	Integer. Cursor from a previous page.
all_pages	Logical. Follow the cursor through every page.

Value

A tibble with one row per game: vgi_id, platform, steam_app_id, north_america, europe, asia, south_america, oceania, africa, middle_east (percentages) and dominant_region. The attribute next_cursor carries the cursor for the next page.

Examples

```
## Not run:
vgi_all_games_regions(steam_app_ids = 4019220)

## End(Not run)
```

---

vgi_all_games_top_countries

Get Top Countries for Many Games (v4)

Description

Retrieve the player-share-by-country breakdown for many games from the v4 /player-insights/games/top-countries endpoint. Games can be selected by Steam App ID, VGI ID or slug; without identifiers the catalogue is paged from the cursor. For one Steam game use vgi_top_countries().

Usage

```
vgi_all_games_top_countries(
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list(),
  steam_app_ids = NULL,
  vgi_ids = NULL,
  slugs = NULL,
  limit = NULL,
  cursor = NULL,
  all_pages = FALSE
)
```

Arguments

auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.
steam_app_ids	Integer vector. Steam App IDs to select. Optional.
vgi_ids	Integer vector. VGI internal game IDs to select. Optional.
slugs	Character vector. VGI game slugs to select. Optional.
limit	Integer. Games per page (API default 200, maximum 1000).
cursor	Integer. Cursor from a previous page.
all_pages	Logical. Follow the cursor through every page.

Value

A tibble with one row per game: vgi_id, platform, steam_app_id, top_countries (list-column of tibbles with country, country_name, percentage, rank), country_count, top_country, top_country_pct. The attribute next_cursor carries the cursor for the next page.

Examples

```
## Not run:
tc <- vgi_all_games_top_countries(steam_app_ids = c(4019220, 730))
tc$top_countries[[1]]

## End(Not run)
```

---

vgi_all_games_wishlist_countries

Get Top Wishlist Countries for Many Games (v4)

Description

Retrieve the wishlist-share-by-country breakdown for many games from the v4 /player-insights/games/top-wishlist-c endpoint. For one Steam game use vgi_top_wishlist_countries().

Usage

```
vgi_all_games_wishlist_countries(
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list(),
  steam_app_ids = NULL,
  vgi_ids = NULL,
  slugs = NULL,
  limit = NULL,
  cursor = NULL,
  all_pages = FALSE
)
```

Arguments

auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.
steam_app_ids	Integer vector. Steam App IDs to select. Optional.
vgi_ids	Integer vector. VGI internal game IDs to select. Optional.
slugs	Character vector. VGI game slugs to select. Optional.
limit	Integer. Games per page (API default 200, maximum 1000).
cursor	Integer. Cursor from a previous page.
all_pages	Logical. Follow the cursor through every page.

Value

A tibble with one row per game: vgi_id, platform, steam_app_id, top_wishlist_countries (list-column of tibbles with country, country_name, percentage, rank), wishlist_country_count, top_wishlist_country, top_wishlist_country_pct. The attribute next_cursor carries the cursor for the next page.

Examples

```
## Not run:
vgi_all_games_wishlist_countries(slugs = "dressmaker")

## End(Not run)
```

---

vgi_all_publisher_games

Get Game IDs for Publishers

Description

Retrieve the VGI game IDs attached to each publisher from `/companies/publishers/game-ids`. Games on every platform are included; use `vgi_publisher_games()` for one publisher's Steam App IDs.

Usage

```
vgi_all_publisher_games(
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list(),
  vgi_ids = NULL,
  slugs = NULL,
  limit = NULL,
  cursor = NULL,
  all_pages = FALSE
)
```

Arguments

`auth_token`	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
`headers`	List. Optional custom headers to include in the API request.
`vgi_ids`	Integer vector. VGI company IDs to select.
`slugs`	Character vector. VGI company slugs to select.
`limit`	Integer. Records per page (maximum 1000).
`cursor`	Integer. Cursor from a previous page.
`all_pages`	Logical. Fetch every page.

Value

A tibble with columns `publisher_id`, `name`, `game_ids` (list-column of VGI game IDs) and `game_count`, sorted by game count. The attribute `next_cursor` carries the cursor for the next page.

Examples

```
## Not run:
vgi_all_publisher_games(vgi_ids = 28663)

## End(Not run)
```

vgi_auth_check *Check VGI API Authentication*

Description

Performs a lightweight authenticated request to verify that the configured credentials can access the API. Returns TRUE on success; otherwise FALSE with an informative message.

Usage

```
vgi_auth_check(auth_token = Sys.getenv("VGI_AUTH_TOKEN"))
```

Arguments

auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.

Value

Logical indicating whether authentication appears valid.

vgi_build_cache *Build Complete Game Cache*

Description

Fetches all games from the API and caches them locally with metadata. This is a one-time operation that should be run periodically to update the cache.

Usage

```
vgi_build_cache(
  force_refresh = FALSE,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN")
)
```

Arguments

force_refresh	Logical. Force refresh even if cache exists. Default FALSE.
auth_token	Character string. Your VGI API authentication token.

Value

Invisible TRUE on success

Examples

```
## Not run:
# Build initial cache
vgi_build_cache()

# Force refresh of cache
vgi_build_cache(force_refresh = TRUE)

## End(Not run)
```

`vgi_cache_stats` *Get Cache Statistics*

Description

Display information about the current cache.

Usage

```
vgi_cache_stats()
```

Value

Invisible list with cache statistics

`vgi_clear_cache` *Clear VGI Cache*

Description

Remove the cached game database.

Usage

```
vgi_clear_cache(confirm = TRUE)
```

Arguments

`confirm`	Logical. Require confirmation. Default TRUE.

Value

Invisible TRUE

vgi_concurrent_players_by_date *Get Concurrent Players Data by Date*

Description

Retrieve concurrent player counts for all games on a specific date, providing a snapshot of active player engagement across the market.

Usage

```
vgi_concurrent_players_by_date(
  date,
  steam_app_ids = NULL,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

date	Character string or Date. The date for which to retrieve data in "YYYY-MM-DD" format.
steam_app_ids	Numeric vector. Optional Steam App IDs to filter results. If not provided, returns data for all available games.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Details

Concurrent player data is one of the most important engagement metrics:

- Real-time indicator of game health
- Direct measure of active player base
- Useful for identifying trending games
- Critical for multiplayer game analysis
- Key metric for seasonal event planning

Peak concurrent typically occurs during prime gaming hours, while average provides a more stable engagement metric.

Value

A data frame with columns:

steamAppId Integer. The Steam App ID

date Character. The date of the data

peakConcurrent Integer. Peak concurrent players on this date

avgConcurrent Integer. Average concurrent players on this date

concurrentRank Integer. Rank by peak concurrent players

Examples

```
## Not run:
# Get concurrent player data
ccu_data <- vgi_concurrent_players_by_date("2024-01-15")

# Top 20 games by concurrent players
top_ccu <- head(ccu_data, 20)
cat("Top 20 games by peak concurrent players:\n")
print(top_ccu[, c("steamAppId", "peakConcurrent", "avgConcurrent")])

# Calculate peak-to-average ratio (indicates play pattern)
ccu_data$peak_avg_ratio <- ccu_data$peakConcurrent /
                           (ccu_data$avgConcurrent + 1)

# Games with spiky play patterns (high peak/avg ratio)
spiky_games <- ccu_data[ccu_data$peak_avg_ratio > 3 &
                        ccu_data$peakConcurrent > 1000, ]
cat("Games with concentrated play times:", nrow(spiky_games), "\n")

# Compare weekend vs weekday
is_weekend <- weekdays(as.Date("2024-01-15")) %in% c("Saturday", "Sunday")
if (!is_weekend) {
  # Get previous weekend data
  last_saturday <- as.Date("2024-01-15") -
                   ((as.numeric(format(as.Date("2024-01-15"), "%w")) + 1) %% 7)
  weekend_ccu <- vgi_concurrent_players_by_date(as.character(last_saturday))

  comparison <- merge(ccu_data, weekend_ccu,
                      by = "steamAppId",
                      suffixes = c("_weekday", "_weekend"))

  # Calculate weekend boost
  comparison$weekend_boost <- (comparison$peakConcurrent_weekend -
                               comparison$peakConcurrent_weekday) /
                              comparison$peakConcurrent_weekday * 100

  # Games with biggest weekend boost
  weekend_games <- comparison[comparison$weekend_boost > 50 &
                              comparison$peakConcurrent_weekday > 500, ]
  cat("Games with >50% weekend boost:", nrow(weekend_games), "\n")
}
```

```
# Analyze player concentration
total_players <- sum(ccu_data$peakConcurrent)
top_10_players <- sum(head(ccu_data$peakConcurrent, 10))
concentration <- (top_10_players / total_players) * 100

cat(sprintf("Top 10 games account for %.1f%% of all concurrent players\n",
            concentration))

# Distribution analysis
hist(log10(ccu_data$peakConcurrent + 1),
     breaks = 40,
     main = "Distribution of Peak Concurrent Players (log scale)",
     xlab = "Log10(Peak CCU + 1)",
     col = "orange")

# Find multiplayer vs single-player patterns
# (Multiplayer games typically have higher avg/peak ratios)
ccu_data$avg_peak_ratio <- ccu_data$avgConcurrent /
                           (ccu_data$peakConcurrent + 1)
likely_multiplayer <- ccu_data[ccu_data$avg_peak_ratio > 0.6 &
                               ccu_data$peakConcurrent > 1000, ]
cat("Likely multiplayer games (high avg/peak):", nrow(likely_multiplayer), "\n")

## End(Not run)
```

`vgi_developers_overview` *Get Developer Overview (v4, multi-platform)*

Description

Retrieve developer summaries from the v4 `/companies/developers` endpoint, including per-platform revenue, release counts and genre breakdowns.

Usage

```
vgi_developers_overview(
  vgi_ids = NULL,
  slugs = NULL,
  cursor = NULL,
  limit = 100,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list(),
  all_pages = FALSE
)
```

Arguments

vgi_ids	Integer vector. VGI company IDs to select. Optional.
slugs	Character vector. VGI company slugs to select. Optional.
cursor	Integer. Cursor from a previous page.
limit	Integer. Records per page (API default 200, maximum 1000).
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.
all_pages	Logical. Follow the cursor through every page.

Value

A tibble with one row per publisher: vgi_company_id, name, slug, classification, country, vgi_url and the flattened platform_data.<platform>.<metric> columns (for example platform_data.steam.revenue_ platform_data.steam.genre_breakdown.rpg). The attribute next_cursor carries the cursor for the next page.

Examples

```
## Not run:
vgi_developers_overview(slugs = "free-lives")

## End(Not run)
```

---

vgi_developer_games *Get a Developer's Steam Games*

Description

Retrieve the Steam App IDs of the games a company develops. The VGI game IDs from /companies/developers/game-ids are resolved to Steam App IDs through /games/metadata; games that are not on Steam are dropped. For the VGI IDs of games on every platform use vgi_all_developer_games().

Usage

```
vgi_developer_games(
  company_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

company_id	Integer. The VGI company ID of the developer.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

An integer vector of Steam App IDs, sorted (empty when none).

Examples

```
## Not run:
vgi_developer_games(28663)

## End(Not run)
```

`vgi_developer_info` *Get Developer Information*

Description

Retrieve a developer's summary from `/companies/developers`. Revenue and game counts are VGI's Steam figures; release counts are given per platform. For many developers or the full per-platform breakdown use `vgi_developers_overview()`.

Usage

```
vgi_developer_info(
  company_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

`company_id`	Integer. The VGI company ID of the developer.
`auth_token`	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
`headers`	List. Optional custom headers to include in the API request.

Value

A one-row tibble with columns `company_id`, `name`, `slug`, `classification`, `country`, `games_developed`, `games_in_development`, `revenue_total`, `revenue_avg_per_game`, `revenue_median_per_game`, `games_released_steam`, `games_released_playstation`, `games_released_xbox` and `vgi_url`. Empty when the developer is unknown.

Examples

```
## Not run:
vgi_developer_info(28663)

## End(Not run)
```

`vgi_developer_list` *Get Developer Names and IDs*

Description

Retrieve the developer directory from `/companies/developers/list`, which pages with a cursor and includes slugs.

Usage

```
vgi_developer_list(
  search = NULL,
  limit = NULL,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list(),
  vgi_ids = NULL,
  slugs = NULL,
  cursor = NULL,
  all_pages = FALSE
)
```

Arguments

`search`	Character. Optional case-insensitive regular expression applied to publisher names after download.
`limit`	Integer. Records per page (maximum 1000).
`auth_token`	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
`headers`	List. Optional custom headers to include in the API request.
`vgi_ids`	Integer vector. VGI company IDs to select.
`slugs`	Character vector. VGI company slugs to select.
`cursor`	Integer. Cursor from a previous page.
`all_pages`	Logical. Follow the cursor through every page.

Value

A tibble with columns `id`, `name`, `slug` and `vgi_url`, sorted by name. The attribute `next_cursor` carries the cursor for the next page.

Examples

```
## Not run:
vgi_developer_list(search = "free lives")

## End(Not run)
```

vgi_fetch_all_games *Smart Fetch Functions for VGI API*

Description

These functions handle pagination and ID-based fetching intelligently Fetch All Games with Pagination

Usage

```
vgi_fetch_all_games(
  cache_results = TRUE,
  verbose = TRUE,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN")
)
```

Arguments

cache_results	Logical. Cache results to disk. Default TRUE.
verbose	Logical. Print progress messages. Default TRUE.
auth_token	Character string. Your VGI API authentication token.

Details

Fetches all games from the API by handling pagination automatically. Results are cached to avoid repeated API calls.

Value

Data frame with all games

vgi_fetch_by_ids *Fetch Games by ID List*

Description

Fetch game data for a list of Steam App IDs: metadata from `/games/metadata`, or one day's active players from `/historical-data`.

Usage

```
vgi_fetch_by_ids(
  steam_app_ids,
  data_type = c("metadata", "active_players"),
  date = NULL,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN")
)
```

Arguments

steam_app_ids	Integer vector. List of Steam App IDs to fetch.
data_type	Character. Type of data to fetch: "metadata" or "active_players".
date	Character or Date. Required for "active_players".
auth_token	Character string. Your VGI API authentication token.

Value

A tibble with the requested game data.

Examples

```
## Not run:
games <- vgi_fetch_by_ids(c(730, 578080, 1172470), "metadata")
active <- vgi_fetch_by_ids(c(730, 578080), "active_players", date = "2025-07-26")

## End(Not run)
```

---

vgi_filter_genre *Filter Games by Genre*

Description

Filter cached games by genre without hitting the API. VGI assigns each game one genre (for example "Shooter"); Steam's own genres are in steam_genres and steam_subgenres. source picks which to match.

Usage

```
vgi_filter_genre(genre, cache_df = NULL, source = c("vgi", "steam"))
```

Arguments

genre	Character string. Regular expression matched case-insensitively (e.g. "Shooter", "RPG").
cache_df	Optional pre-loaded cache. If NULL, will load automatically.
source	"vgi" (default) matches VGI's genre and subgenre; "steam" matches steam_genres and steam_subgenres.

Value

A tibble of the matching games.

Examples

```
## Not run:
shooters <- vgi_filter_genre("Shooter")
steam_shooters <- vgi_filter_genre("Shooter", source = "steam")

## End(Not run)
```

`vgi_followers_by_date` *Get Followers Data by Date*

Description

Retrieve follower counts for all games on a specific date, useful for tracking market-wide community engagement trends.

Usage

```
vgi_followers_by_date(
  date,
  steam_app_ids = NULL,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

`date`	Character string or Date. The date for which to retrieve data in "YYYY-MM-DD" format.
`steam_app_ids`	Numeric vector. Optional. Steam App IDs to filter results. If not provided, returns data for all available games.
`auth_token`	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
`headers`	List. Optional custom headers to include in the API request.

Details

Steam followers represent users who want to stay updated about a game. This metric is valuable for:

- Measuring long-term community engagement
- Tracking marketing campaign effectiveness
- Identifying games with growing communities
- Benchmarking community size across genres
- Early warning for declining interest

Unlike wishlists, followers persist after game purchase, making this a good metric for both released and unreleased games.

Value

A data frame with columns:

steamAppId Integer. The Steam App ID

date Character. The date of the data

followerCount Integer. Number of followers

followerRank Integer. Rank by follower count

Examples

```
## Not run:
# Get follower data for a specific date
followers <- vgi_followers_by_date("2024-01-15")

# Top 20 most followed games
top_followed <- head(followers, 20)
print(top_followed)

# Compare with wishlist data to find engagement patterns
wishlists <- vgi_wishlists_by_date("2024-01-15")
engagement <- merge(followers, wishlists, by = "steamAppId")
engagement$follow_to_wishlist_ratio <-
  engagement$followerCount / (engagement$wishlistCount + 1)

# Games with high follower/wishlist ratio (strong community)
high_engagement <- engagement[engagement$follow_to_wishlist_ratio > 2 &
                              engagement$followerCount > 10000, ]
cat("High community engagement games:", nrow(high_engagement), "\n")

# Monthly follower growth analysis
month_ago <- as.Date("2024-01-15") - 30
followers_prev <- vgi_followers_by_date(as.character(month_ago))

growth <- merge(followers, followers_prev,
                by = "steamAppId",
                suffixes = c("_now", "_prev"))
growth$monthly_change <- growth$followerCount_now - growth$followerCount_prev
growth$monthly_pct <- (growth$monthly_change / growth$followerCount_prev) * 100

# Fastest growing communities (min 5000 followers)
qualified <- growth[growth$followerCount_prev >= 5000, ]
fastest <- head(qualified[order(-qualified$monthly_pct), ], 20)

cat("Fastest growing communities (>5000 followers):\n")
print(fastest[, c("steamAppId", "followerCount_now",
                  "monthly_change", "monthly_pct")])

# Find games losing followers
declining <- growth[growth$monthly_change < -500, ]
cat("Games losing 500+ followers this month:", nrow(declining), "\n")
```

```
# Analyze follower distribution by tier
follower_tiers <- cut(followers$followerCount,
                      breaks = c(0, 1000, 10000, 50000, 100000, Inf),
                      labels = c("<1K", "1K-10K", "10K-50K", "50K-100K", ">100K"))
tier_summary <- table(follower_tiers)

barplot(tier_summary,
        main = "Distribution of Games by Follower Count",
        xlab = "Follower Tier",
        ylab = "Number of Games",
        col = "skyblue")

## End(Not run)
```

`vgi_games_metadata` *Get Game Metadata for Many Games (v4, multi-platform)*

Description

Retrieve metadata from the v4 `/games/metadata` endpoint. Games can be selected by Steam App ID, VGI ID or slug, and the endpoint pages with a cursor. Without any identifiers it lists the catalogue from the cursor.

Usage

```
vgi_games_metadata(
  steam_app_ids = NULL,
  vgi_ids = NULL,
  slugs = NULL,
  limit = NULL,
  cursor = NULL,
  all_pages = FALSE,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

`steam_app_ids`	Integer vector. Steam App IDs to select. Optional.
`vgi_ids`	Integer vector. VGI internal game IDs to select. Optional.
`slugs`	Character vector. VGI game slugs to select. Optional.
`limit`	Integer. Games per page (API default 200, maximum 1000).
`cursor`	Integer. VGI ID after which results start (from the `next_cursor` attribute of a previous page).
`all_pages`	Logical. Follow `nextCursor` until every page is fetched.
`auth_token`	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
`headers`	List. Optional custom headers to include in the API request.

Value

A tibble with one row per game: vgi_id, slug, name, steam_app_id (parsed from the Steam store URL), per-platform columns price_steam, price_xbox, price_playstation, release_date_steam, release_date_xbox, release_date_playstation, store_url_steam, store_url_xbox, store_url_playstation, plus steam_full_release_date, genre, subgenre, steam_genres, steam_subgenres, languages, publisher_classification, publishing_type, vgi_url, publisher_id, publisher_name, developer_id, developer_name, and list-columns platforms, game_engines, steam_tags, themes, art_styles, character, game_modes, developers, publishers. The attribute next_cursor carries the cursor for the next page (NULL when exhausted).

Examples

```
## Not run:
vgi_games_metadata(steam_app_ids = 4019220)
vgi_games_metadata(slugs = "dressmaker")$vgi_id

# Page through the catalogue 500 games at a time
page1 <- vgi_games_metadata(limit = 500)
page2 <- vgi_games_metadata(limit = 500, cursor = attr(page1, "next_cursor"))

## End(Not run)
```

---

vgi_game_list *Get the Full Game List*

Description

Retrieve the catalogue of games known to VGI, on every platform, from /games/game-list. The response is large (several MB) and unpaginated; consider caching the result.

Usage

```
vgi_game_list(auth_token = Sys.getenv("VGI_AUTH_TOKEN"), headers = list())
```

Arguments

auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

A tibble with columns vgi_id, steam_app_id (NA for games that are not on Steam) and name, sorted by VGI ID.

Examples

```
## Not run:
games <- vgi_game_list()
nrow(games)

## End(Not run)
```

`vgi_game_metadata` *Get Game Metadata for a Steam Game*

Description

Retrieves the metadata of one Steam game from `/games/metadata`. The row has the multi-platform shape of `vgi_games_metadata()`: VGI's own `genre` and `subgenre`, Steam's `steam_genres` and `steam_subgenres`, per-platform prices, release dates and store URLs, plus list-columns for tags, modes, themes, developers and publishers.

Usage

```
vgi_game_metadata(
  steam_app_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

`steam_app_id`	Character or numeric. The Steam App ID of the game.
`auth_token`	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
`headers`	List. Optional custom headers to include in the API request.

Value

A one-row tibble with the columns of `vgi_games_metadata()`. An empty tibble is returned when the API has no such game.

Examples

```
## Not run:
vgi_game_metadata(4019220)$name

## End(Not run)
```

`vgi_game_metadata_batch` *Get Metadata for Several Steam Games*

Description

Retrieves metadata for multiple Steam games from `/games/metadata`, up to 500 IDs per request. When the API rejects a request because one ID is unknown, that chunk is fetched one game at a time.

Usage

```
vgi_game_metadata_batch(
  steam_app_ids,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

`steam_app_ids`	Numeric vector. The Steam App IDs of the games.
`auth_token`	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
`headers`	List. Optional custom headers to include in the API request.

Value

A tibble with one row per game found, in the shape of `vgi_games_metadata()`. IDs the API does not know are reported in a warning.

Examples

```
## Not run:
vgi_game_metadata_batch(c(892970, 1245620, 105600))

## End(Not run)
```

vgi_game_summary *Get Comprehensive Game Summary*

Description

Retrieve all available metrics for specified games over a date range. This function aggregates data from multiple VGI endpoints to provide a complete overview of game performance.

Usage

```
vgi_game_summary(
  steam_app_ids,
  start_date,
  end_date = NULL,
  metrics = c("concurrent", "active", "revenue", "units"),
  game_names = NULL,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  batch_size = NULL,
  batch_delay = NULL
)
```

Arguments

steam_app_ids	Numeric vector. Steam App IDs to analyze.
start_date	Character string or Date. Start date for the analysis.
end_date	Character string or Date. End date for the analysis. Defaults to start_date if not provided (single day analysis).
metrics	Character vector. Which metrics to retrieve. Defaults to all. Options: "concurrent", "active", "revenue", "units", "reviews", "followers", "wishlists", "price", "metadata"
game_names	Character vector. Optional game names corresponding to steam_app_ids. If not provided, names will be fetched from metadata.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
batch_size	Integer. Number of API calls per batch before pausing. Defaults to automatic calculation based on date range.
batch_delay	Numeric. Seconds to pause between batches. Defaults to 1 second. Set to 0 to disable rate limiting.

Details

This function makes multiple API calls to gather comprehensive data. For efficiency, it uses the steamAppIds parameter where supported.

Note: Some endpoints may have data availability limitations:

- Active players: DAU from 2024-03-18, MAU from 2024-03-23
- Some metrics may not be available for all games

Value

A list containing:

summary_table Aggregated metrics for each game

time_series List of time series data frames by metric

metadata Game metadata if requested

api_calls Number of API calls made

errors Any errors encountered during data collection

Examples

```
## Not run:
# Analyze a single game for one week
bf5_summary <- vgi_game_summary(
  steam_app_ids = 1238810,  # Battlefield V
  start_date = "2024-07-01",
  end_date = "2024-07-07"
)

# Analyze multiple games
battlefield_summary <- vgi_game_summary(
  steam_app_ids = c(1517290, 1238810),  # BF2042 and BFV
  start_date = "2024-07-01",
  end_date = "2024-07-07",
  metrics = c("concurrent", "active", "revenue", "units")
)

# View summary table
print(battlefield_summary$summary_table)

# Plot time series
ggplot(battlefield_summary$time_series$concurrent,
       aes(x = date, y = peakConcurrent, color = factor(steamAppId))) +
  geom_line() +
  labs(title = "Peak CCU Over Time")

## End(Not run)
```

---

vgi_game_summary_yoy *Get Year-over-Year Game Summary Comparison*

Description

Compare game metrics across multiple years for the same time period. Supports flexible date specification using months or explicit dates.

Usage

```
vgi_game_summary_yoy(
  steam_app_ids,
  years,
  start_month = NULL,
  end_month = NULL,
  start_date = NULL,
  end_date = NULL,
  metrics = c("concurrent", "revenue", "units"),
  include_growth = TRUE,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  batch_size = NULL,
  batch_delay = NULL
)
```

Arguments

steam_app_ids	Numeric vector. Steam App IDs to analyze.
years	Numeric vector. Years to compare (e.g., c(2023, 2024, 2025)).
start_month	Character string. Start month name or abbreviation (e.g., "January", "Jan", "1"). Ignored if start_date is provided.
end_month	Character string. End month name or abbreviation. Defaults to start_month if not provided. Ignored if end_date is provided.
start_date	Character string or Date. Explicit start date in "MM-DD" or "YYYY-MM-DD" format. Takes precedence over start_month.
end_date	Character string or Date. Explicit end date. Takes precedence over end_month.
metrics	Character vector. Which metrics to retrieve. Defaults to c("concurrent", "revenue", "units").
include_growth	Logical. Whether to calculate year-over-year growth percentages. Defaults to TRUE.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
batch_size	Integer. Number of API calls per batch before pausing. Defaults to automatic calculation based on date range.
batch_delay	Numeric. Seconds to pause between batches. Defaults to value from VGI_BATCH_DELAY environment variable or 1 second.

Details

The function supports two ways to specify the comparison period:

1. **Month-based**: Specify start_month and optionally end_month. The function will use these months for each year in the years parameter.
2. **Date-based**: Specify start_date and end_date with explicit month/day (e.g., "03-15" for March 15). These will be applied to each year.

Year-over-year growth is calculated as: ((current - previous) / previous) * 100

Value

A vgi_yoy_comparison object (classed list) containing:

comparison_table Data frame of per-game metrics by year, with optional year-over-year growth columns.

yearly_summaries Named list of full vgi_game_summary() outputs, one entry per year in the comparison.

time_series_comparison Named list of combined time-series data frames (e.g., concurrent, revenue, units) with normalized dates for cross-year plotting.

period Human-readable description of the comparison window.

years Sorted numeric vector of years that were compared.

api_calls Total number of API calls made across all years.

Examples

```
## Not run:
# Compare Q1 performance across years
q1_comparison <- vgi_game_summary_yoy(
  steam_app_ids = c(892970, 1145360),
  years = c(2023, 2024, 2025),
  start_month = "Jan",
  end_month = "Mar"
)

# Compare specific date ranges
holiday_comparison <- vgi_game_summary_yoy(
  steam_app_ids = 892970,
  years = c(2022, 2023, 2024),
  start_date = "11-15",  # Nov 15
  end_date = "01-15"     # Jan 15 (crosses year boundary)
)

# Compare full years
yearly_comparison <- vgi_game_summary_yoy(
  steam_app_ids = c(1517290, 1238810),
  years = c(2023, 2024),
  start_month = "January",
  end_month = "December"
)

## End(Not run)
```

---

vgi_get_games_by_id *Get Games by Steam App IDs*

Description

Retrieve game information for specific Steam App IDs from the cache, fetching any missing games from /games/metadata.

Usage

```
vgi_get_games_by_id(steam_app_ids, cache_df = NULL, fetch_missing = TRUE)
```

Arguments

steam_app_ids	Integer vector. Steam App IDs to retrieve.
cache_df	Optional pre-loaded cache. If NULL, will load automatically.
fetch_missing	Logical. Fetch from API if not in cache. Default TRUE.

Value

A tibble with one row per game found.

Examples

```
## Not run:
games <- vgi_get_games_by_id(c(730, 578080, 1172470))

## End(Not run)
```

---

vgi_historical_data *Get Full Historical Data for a Game*

Description

Retrieve the complete daily history for a single Steam game and split it into tidy per-metric time series.

Usage

```
vgi_historical_data(
  steam_app_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

steam_app_id	Integer. The Steam App ID of the game.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Details

For a single-date snapshot across many games (including the v4 multi-platform, country and region breakdowns) use vgi_historical_data_by_date().

Value

A list with element steam_app_id plus one tibble (or NULL when the API has no data for that metric) for each of:

revenue date, revenue (cumulative total), daily_revenue; v4 premiumRevenue* fields

units_sold date, units_sold (cumulative total), daily_units; v4 unitsOwned* fields

concurrent_players date, ccu_avg, ccu_median, ccu_max, ccu_min

active_players date, dau, mau

reviews date, positive, negative, positive_change, negative_change

wishlists date, wishlists, wishlists_change

followers date, followers, followers_change

price_history date, price_initial, price_final

The daily element holds the raw flat daily table with every metric column returned by the API (snake_case).

Examples

```
## Not run:
hist <- vgi_historical_data(4019220)
hist$concurrent_players
hist$revenue

## End(Not run)
```

---

vgi_historical_data_by_date

Get a Historical Data Snapshot for a Date (v4)

Description

Retrieve one day of metrics for many games at once from the v4 /historical-data endpoint. The v4 endpoint is multi-platform and can break the numbers down by country or region.

Usage

```
vgi_historical_data_by_date(
  date,
  steam_app_ids = NULL,
  vgi_ids = NULL,
  slugs = NULL,
  countries = NULL,
  regions = NULL,
  limit = NULL,
  cursor = NULL,
  all_pages = FALSE,
```

```
  platforms = NULL,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

date	Character string or Date. Snapshot date (YYYY-MM-DD). Earliest supported value is 2014-01-01. NULL lets the API choose its default date.
steam_app_ids	Integer vector. Steam App IDs to select. Optional.
vgi_ids	Integer vector. VGI internal game IDs to select. Optional.
slugs	Character vector. VGI game slugs to select. Optional.
countries	Character vector of 2-letter ISO country codes. When supplied the API returns one record per game and country.
regions	Character vector of VGI region slugs. When supplied the API returns one record per game and region.
limit	Integer. Games per page (API default 5, maximum 1000).
cursor	Integer. Cursor from a previous page (next_cursor attribute).
all_pages	Logical. Follow nextCursor until every page is fetched.
platforms	Character vector. Keep only these platforms ("steam", "xbox", "playstation"). NULL keeps every platform.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

A tibble with one row per game (and platform / country / region when requested). Columns are the snake_case API fields, e.g. vgi_id, platform, external_id, steam_app_id (NA for non-steam rows), date, country, revenue_total, revenue_change, units_sold_total, dau, mau, wishlists_total, ccu_max, price_final. The attribute next_cursor carries the cursor for the next page (NULL when exhausted).

Examples

```
## Not run:
snap <- vgi_historical_data_by_date("2026-09-28", steam_app_ids = 4019220)
snap[, c("steam_app_id", "ccu_max", "revenue_total")]

# Country breakdown
vgi_historical_data_by_date("2026-09-28", steam_app_ids = 4019220,
                            countries = c("US", "DE"))

## End(Not run)
```

`vgi_insights_ccu` *Get Concurrent Users (CCU) Data*

Description

Retrieve the daily concurrent player history for a single Steam game.

Usage

```
vgi_insights_ccu(
  steam_app_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

`steam_app_id`	Integer. The Steam App ID of the game.
`auth_token`	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
`headers`	List. Optional custom headers to include in the API request.

Value

A list containing:

steam_app_id Integer. The Steam App ID

player_history Tibble with columns `date`, `avg`, `median`, `max`, `min` (concurrent players), sorted by date

Examples

```
## Not run:
ccu <- vgi_insights_ccu(steam_app_id = 4019220)
plot(ccu$player_history$date, ccu$player_history$max, type = "l")

## End(Not run)
```

`vgi_insights_dau_mau` *Get Daily and Monthly Active Users (DAU/MAU)*

Description

Retrieve the active player history for a single Steam game.

Usage

```
vgi_insights_dau_mau(
  steam_app_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

`steam_app_id`	Integer. The Steam App ID of the game.
`auth_token`	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
`headers`	List. Optional custom headers to include in the API request.

Details

DAU is available from 2024-03-18 and MAU from 2024-03-23.

Value

A list containing:

steam_app_id Integer. The Steam App ID

player_history Tibble with columns `date`, `dau`, `mau`

Examples

```
## Not run:
active <- vgi_insights_dau_mau(steam_app_id = 4019220)
tail(active$player_history)

## End(Not run)
```

`vgi_insights_followers` *Get Follower History for a Game*

Description

Retrieve the Steam follower history for a single game.

Usage

```
vgi_insights_followers(
  steam_app_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

`steam_app_id`	Integer. The Steam App ID of the game.
`auth_token`	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
`headers`	List. Optional custom headers to include in the API request.

Value

A list containing:

steam_app_id Integer. The Steam App ID

followers_change Tibble with columns `date`, `followers_total`, `followers_change`

Examples

```
## Not run:
fol <- vgi_insights_followers(steam_app_id = 4019220)
tail(fol$followers_change)

## End(Not run)
```

vgi_insights_player_regions *Get Player Region Distribution for a Game*

Description

Retrieve the share of players by world region for a single Steam game.

Usage

```
vgi_insights_player_regions(
  steam_app_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

steam_app_id	Integer. The Steam App ID of the game.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

A list containing steam_app_id and regions, a tibble with columns region_name, rank and percentage, sorted by rank.

See Also

vgi_top_regions() returns just the regions tibble.

Examples

```
## Not run:
vgi_insights_player_regions(steam_app_id = 4019220)$regions

## End(Not run)
```

vgi_insights_playtime *Get Playtime Insights for a Game*

Description

Retrieve playtime statistics for a single Steam game. For many games at once, or for country / region filtered playtime, use vgi_all_games_playtime() (v4).

Usage

```
vgi_insights_playtime(
  steam_app_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

steam_app_id	Integer. The Steam App ID of the game.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

A list containing steam_app_id, avg_playtime and median_playtime (minutes), avg_playtime_rank, avg_playtime_prct, and playtime_ranges, a tibble with columns range and percentage.

Examples

```
## Not run:
pt <- vgi_insights_playtime(steam_app_id = 4019220)
pt$median_playtime
pt$playtime_ranges

## End(Not run)
```

vgi_insights_price_history *Get Price History Data for a Game*

Description

Retrieve the Steam price-change periods for a game: the Steam rows of vgi_price_history(), which also covers PlayStation and Xbox prices and VGI-id or slug lookup.

Usage

```
vgi_insights_price_history(
  steam_app_id,
  currency = NULL,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

steam_app_id	Integer. The Steam App ID of the game.
currency	Character. Optional ISO currency code (e.g. "USD", "EUR"). When omitted, price periods for every currency are returned.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

A list containing:

steam_app_id Integer. The Steam App ID

currency Character. The requested currency, or "ALL"

price_changes Tibble with columns currency, price_initial, price_final, first_date, last_date (NA for the current period), newest period first

Examples

```
## Not run:
usd <- vgi_insights_price_history(4019220, currency = "USD")
usd$price_changes

all_prices <- vgi_insights_price_history(4019220)
table(all_prices$price_changes$currency)

## End(Not run)
```

---

vgi_insights_revenue *Get Revenue History for a Game*

Description

Retrieve the daily revenue history for a single Steam game. In v4 the figures come from premiumRevenueChange / premiumRevenueTotal (the successors of the deprecated revenueChange / revenueTotal).

Usage

```
vgi_insights_revenue(
  steam_app_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

steam_app_id	Integer. The Steam App ID of the game.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

A tibble with columns:

steam_app_id Integer. The Steam App ID

date Date

revenue_change Numeric. Revenue earned on that day (USD)

revenue_total Numeric. Cumulative revenue to date (USD)

Examples

```
## Not run:
rev <- vgi_insights_revenue(steam_app_id = 4019220)
plot(rev$date, rev$revenue_change, type = "l")

## End(Not run)
```

---

vgi_insights_reviews *Get Review History for a Game*

Description

Retrieve the daily Steam review history for a single game.

Usage

```
vgi_insights_reviews(
  steam_app_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

steam_app_id	Integer. The Steam App ID of the game.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

A tibble with columns steam_app_id, date, positive, negative (cumulative totals), total, positive_ratio, positive_change and negative_change (reviews added that day).

Examples

```
## Not run:
reviews <- vgi_insights_reviews(steam_app_id = 4019220)
tail(reviews[, c("date", "positive", "negative", "positive_ratio")])

## End(Not run)
```

---

vgi_insights_units *Get Units Sold History for a Game*

Description

Retrieve the daily units history for a single Steam game. In v4 the figures come from unitsOwnedChange / unitsOwnedTotal (the successors of the deprecated unitsSoldChange / unitsSoldTotal; they include copies obtained for free).

Usage

```
vgi_insights_units(
  steam_app_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

steam_app_id	Integer. The Steam App ID of the game.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

A tibble with columns:

steam_app_id Integer. The Steam App ID

date Date

units_sold_change Integer. Units sold on that day

units_sold_total Integer. Cumulative units sold to date

Examples

```
## Not run:
units <- vgi_insights_units(steam_app_id = 4019220)
max(units$units_sold_total)

## End(Not run)
```

---

vgi_insights_wishlists

Get Wishlist History for a Game

Description

Retrieve the outstanding-wishlist history for a single Steam game.

Usage

```
vgi_insights_wishlists(
  steam_app_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

steam_app_id	Integer. The Steam App ID of the game.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

A list containing:

steam_app_id Integer. The Steam App ID

wishlist_changes Tibble with columns `date`, `wishlists_total`, `wishlists_change`

Examples

```
## Not run:
wl <- vgi_insights_wishlists(steam_app_id = 4019220)
tail(wl$wishlist_changes)

## End(Not run)
```

`vgi_load_cache` *Load Game Cache*

Description

Loads the cached game database.

Usage

```
vgi_load_cache(auto_build = TRUE)
```

Arguments

`auto_build`	Logical. Automatically build cache if it doesn't exist. Default TRUE.

Value

Data frame with all cached games and metadata

`vgi_peak_ccu_by_ids` *Get peak concurrent players (CCU) for multiple Steam app IDs*

Description

Get peak concurrent players (CCU) for multiple Steam app IDs

Usage

```
vgi_peak_ccu_by_ids(steam_app_ids)
```

Arguments

`steam_app_ids`	Integer vector of Steam app IDs

Value

A tibble with columns: name, steam_app_id, peak_ccu, peak_date

vgi_peak_ccu_by_names *Get peak concurrent players (CCU) for multiple games by names*

Description

Resolves each game name to a Steam app ID using `vgi_search_games(title, limit = 5)` with a simple exact/startsWith prioritization, then fetches peak CCU via `vgi_insights_ccu()` per resolved ID. No all-games endpoints are used.

Usage

```
vgi_peak_ccu_by_names(game_names)
```

Arguments

`game_names`	Character vector of game names (e.g., "THE FINALS", "Halo Infinite")

Value

A tibble with columns: input_name, name, steam_app_id, peak_ccu, peak_date

vgi_player_overlap *Get Player Overlap for a Game*

Description

Retrieve the games whose audiences overlap most with a given game from `/player-overlap`. Identify the game by Steam App ID, VGI ID or slug.

Usage

```
vgi_player_overlap(
  steam_app_id = NULL,
  limit = 10,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list(),
  vgi_id = NULL,
  slug = NULL
)
```

Arguments

steam_app_id	Integer. The Steam App ID of the game.
limit	Integer. Maximum number of overlapping games to return, largest units overlap first.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.
vgi_id	Integer. VGI internal game ID.
slug	Character. VGI game slug.

Value

A list containing:

steam_app_id Integer. The requested game (NA for non-Steam games)

vgi_id Integer. VGI ID of the requested game

player_overlaps Tibble with one row per overlapping game: steam_app_id, vgi_id, median_playtime, units_sold_overlap, units_sold_overlap_percentage, units_sold_overlap_index, mau_overlap, mau_overlap_percentage, mau_overlap_index, wishlist_overlap, wishlist_overlap_percentage, wishlist_overlap_index

Examples

```
## Not run:
ov <- vgi_player_overlap(4019220, limit = 20)
head(ov$player_overlaps)

vgi_player_overlap(slug = "dressmaker")

## End(Not run)
```

---

vgi_price_history *Get Price History for a Game (v4, multi-platform)*

Description

Retrieve price-change periods for a game from the v4 /price-history endpoint. Exactly one of steam_app_id, vgi_id or slug identifies the game. For Steam prices only see vgi_insights_price_history().

Usage

```
vgi_price_history(
  steam_app_id = NULL,
  vgi_id = NULL,
  slug = NULL,
  currency = NULL,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

steam_app_id	Integer. Steam App ID of the game.
vgi_id	Integer. VGI internal game ID.
slug	Character. VGI game slug (e.g. "dressmaker").
currency	Character vector. Optional ISO currency codes to keep (filtering happens client-side; the API returns every currency).
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

A tibble with one row per price period and columns `platform`, `external_id`, `currency`, `price_initial`, `price_final`, `first_date`, `last_date` (NA for the current period).

Examples

```
## Not run:
vgi_price_history(steam_app_id = 4019220, currency = "USD")
vgi_price_history(slug = "dressmaker")

## End(Not run)
```

---

vgi_publishers_overview

Get Publisher Overview (v4, multi-platform)

Description

Retrieve publisher summaries from the v4 `/companies/publishers` endpoint, including per-platform revenue, release counts and genre breakdowns.

Usage

```
vgi_publishers_overview(
  vgi_ids = NULL,
  slugs = NULL,
  cursor = NULL,
  limit = 100,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list(),
  all_pages = FALSE
)
```

Arguments

vgi_ids	Integer vector. VGI company IDs to select. Optional.
slugs	Character vector. VGI company slugs to select. Optional.
cursor	Integer. Cursor from a previous page.
limit	Integer. Records per page (API default 200, maximum 1000).
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.
all_pages	Logical. Follow the cursor through every page.

Value

A tibble with one row per publisher: vgi_company_id, name, slug, classification, country, vgi_url and the flattened platform_data.<platform>.<metric> columns (for example platform_data.steam.revenue_ platform_data.steam.genre_breakdown.rpg). The attribute next_cursor carries the cursor for the next page.

Examples

```
## Not run:
vgi_publishers_overview(vgi_ids = 28663)
vgi_publishers_overview(limit = 500)

## End(Not run)
```

---

vgi_publisher_games *Get a Publisher's Steam Games*

Description

Retrieve the Steam App IDs of the games a company publishes. The VGI game IDs from /companies/publishers/game-id are resolved to Steam App IDs through /games/metadata; games that are not on Steam are dropped. For the VGI IDs of games on every platform use vgi_all_publisher_games().

Usage

```
vgi_publisher_games(
  company_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

company_id	Integer. The VGI company ID of the publisher.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

An integer vector of Steam App IDs, sorted (empty when none).

Examples

```
## Not run:
vgi_publisher_games(28663)

## End(Not run)
```

---

vgi_publisher_info *Get Publisher Information*

Description

Retrieve a publisher's summary from `/companies/publishers`. Revenue and game counts are VGI's Steam figures; release counts are given per platform. For many publishers or the full per-platform breakdown use `vgi_publishers_overview()`.

Usage

```
vgi_publisher_info(
  company_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

company_id	Integer. The VGI company ID of the publisher.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

A one-row tibble with columns company_id, name, slug, classification, country, games_published, games_in_development, revenue_total, revenue_avg_per_game, revenue_median_per_game, games_released_steam, games_released_playstation, games_released_xbox and vgi_url. Empty when the publisher is unknown.

Examples

```
## Not run:
vgi_publisher_info(28663)

## End(Not run)
```

---

vgi_publisher_list *Get Publisher Names and IDs*

Description

Retrieve the publisher directory from /companies/publishers/list, which pages with a cursor and includes slugs.

Usage

```
vgi_publisher_list(
  search = NULL,
  limit = NULL,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list(),
  vgi_ids = NULL,
  slugs = NULL,
  cursor = NULL,
  all_pages = FALSE
)
```

Arguments

search	Character. Optional case-insensitive regular expression applied to publisher names after download.
limit	Integer. Records per page (maximum 1000).
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.
vgi_ids	Integer vector. VGI company IDs to select.
slugs	Character vector. VGI company slugs to select.
cursor	Integer. Cursor from a previous page.
all_pages	Logical. Follow the cursor through every page.

Value

A tibble with columns id, name, slug and vgi_url, sorted by name. The attribute next_cursor carries the cursor for the next page.

Examples

```
## Not run:
vgi_publisher_list(search = "valve")
vgi_publisher_list(slugs = "free-lives")

## End(Not run)
```

---

vgi_revenue_by_date *Get Revenue Data by Date*

Description

Retrieve revenue data for all games on a specific date, providing a comprehensive view of market financial performance.

Usage

```
vgi_revenue_by_date(
  date,
  steam_app_ids = NULL,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

date	Character string or Date. The date for which to retrieve data in "YYYY-MM-DD" format.
steam_app_ids	Numeric vector. Optional. Steam App IDs to filter results. If not provided, returns data for all available games.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Details

Revenue data is the most important commercial metric for:

- Market size estimation
- Financial performance benchmarking
- ROI analysis

- Publisher/developer valuations
- Investment decisions

All revenue figures are in USD and represent gross revenue before platform fees and taxes.

Value

A data frame with columns:

steamAppId Integer. The Steam App ID

date Character. The date of the data

revenue Numeric. Cumulative revenue in USD as of this date

dailyRevenue Numeric. Revenue generated on this specific day

revenueRank Integer. Rank by total revenue

Examples

```
## Not run:
# Get revenue data for a specific date
revenue_data <- vgi_revenue_by_date("2024-01-15")

# Top 20 highest-grossing games
top_revenue <- head(revenue_data, 20)
cat("Top 20 games by revenue:\n")
print(top_revenue[, c("steamAppId", "revenue", "dailyRevenue")])

# Format revenue for display
top_revenue$revenue_millions <- round(top_revenue$revenue / 1000000, 2)
cat("Top game revenue: $", top_revenue$revenue_millions[1], "M\n", sep = "")

# Calculate market concentration
total_revenue <- sum(revenue_data$revenue)
top_10_revenue <- sum(head(revenue_data$revenue, 10))
concentration <- (top_10_revenue / total_revenue) * 100
cat(sprintf("Top 10 games represent %.1f%% of total revenue\n", concentration))

# Daily revenue leaders
prev_date <- as.Date("2024-01-15") - 1
revenue_prev <- vgi_revenue_by_date(as.character(prev_date))

daily_rev <- merge(revenue_data, revenue_prev,
                   by = "steamAppId",
                   suffixes = c("_today", "_yesterday"))
daily_rev$revenue_today <- daily_rev$revenue_today - daily_rev$revenue_yesterday

# Games with highest daily revenue
top_daily_rev <- head(daily_rev[order(-daily_rev$revenue_today), ], 20)
cat("Top daily revenue generators:\n")
top_daily_rev$daily_rev_k <- round(top_daily_rev$revenue_today / 1000, 1)
print(top_daily_rev[, c("steamAppId", "daily_rev_k")])
```

```
# Revenue distribution analysis
revenue_tiers <- cut(revenue_data$revenue,
                     breaks = c(0, 10000, 100000, 1000000, 10000000, Inf),
                     labels = c("<$10K", "$10K-100K", "$100K-1M",
                                "$1M-10M", ">$10M"))
tier_summary <- table(revenue_tiers)

barplot(tier_summary,
        main = "Games by Revenue Tier",
        xlab = "Revenue Tier",
        ylab = "Number of Games",
        col = "gold")

# Year-over-year growth analysis
last_year <- as.Date("2024-01-15") - 365
revenue_ly <- vgi_revenue_by_date(as.character(last_year))

yoy <- merge(revenue_data, revenue_ly,
             by = "steamAppId",
             suffixes = c("_now", "_lastyear"))
yoy$yoy_growth <- ((yoy$revenue_now - yoy$revenue_lastyear) /
                   yoy$revenue_lastyear) * 100

# Fastest growing games by revenue
min_revenue <- 100000  # Only games with substantial revenue
qualified <- yoy[yoy$revenue_lastyear >= min_revenue, ]
fastest_growing <- head(qualified[order(-qualified$yoy_growth), ], 20)

cat("Fastest growing games (YoY revenue):\n")
print(fastest_growing[, c("steamAppId", "revenue_now", "yoy_growth")])

## End(Not run)
```

`vgi_reviews_by_date` *Get Reviews Data by Date*

Description

Retrieve review data for all games on a specific date, useful for market-wide analysis and trend identification.

Usage

```
vgi_reviews_by_date(
  date,
  steam_app_ids = NULL,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

date	Character string or Date. The date for which to retrieve data in "YYYY-MM-DD" format.
steam_app_ids	Numeric vector. Optional. Steam App IDs to filter results. If not provided, returns data for all available games.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Details

This endpoint enables:

- Daily market sentiment analysis
- Identifying games with review bombs or surges
- Tracking industry-wide review trends
- Comparing review activity across multiple games
- Building review trend dashboards

The data represents the cumulative review counts as of the specified date.

Value

A data frame with columns:

steamAppId Integer. The Steam App ID

date Character. The date of the data

positiveReviews Integer. Number of positive reviews

negativeReviews Integer. Number of negative reviews

totalReviews Integer. Total number of reviews

positiveRatio Numeric. Ratio of positive reviews (0-1)

Examples

```
## Not run:
# Get reviews for a specific date
reviews_data <- vgi_reviews_by_date("2024-01-15")

# Find games with most reviews on this date
top_reviewed <- head(reviews_data[order(-reviews_data$totalReviews), ], 20)
print(top_reviewed)

# Find games with best review ratios (min 100 reviews)
qualified <- reviews_data[reviews_data$totalReviews >= 100, ]
best_rated <- head(qualified[order(-qualified$positiveRatio), ], 20)
cat("Best rated games with 100+ reviews:\n")
print(best_rated[, c("steamAppId", "positiveRatio", "totalReviews")])
```

```
# Analyze review distribution
hist(reviews_data$positiveRatio[reviews_data$totalReviews >= 10],
     breaks = 20,
     main = "Distribution of Review Scores",
     xlab = "Positive Review Ratio",
     col = "lightblue")
abline(v = 0.7, col = "green", lwd = 2, lty = 2)
abline(v = 0.5, col = "orange", lwd = 2, lty = 2)

# Compare with previous date
prev_date <- as.Date("2024-01-15") - 1
prev_reviews <- vgi_reviews_by_date(as.character(prev_date))

# Merge to find daily changes
comparison <- merge(reviews_data, prev_reviews,
                    by = "steamAppId",
                    suffixes = c("_today", "_yesterday"))

# Calculate daily review additions
comparison$new_reviews <- comparison$totalReviews_today - comparison$totalReviews_yesterday
comparison$ratio_change <- comparison$positiveRatio_today - comparison$positiveRatio_yesterday

# Find games with biggest review changes
biggest_changes <- head(comparison[order(-abs(comparison$new_reviews)), ], 10)
cat("Games with most review activity:\n")
print(biggest_changes[, c("steamAppId", "new_reviews", "ratio_change")])

## End(Not run)
```

`vgi_search_games` *Search Games in Video Game Insights*

Description

Search for games by title using the Video Game Insights database.

Usage

```
vgi_search_games(
  query,
  limit = 10,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list(),
  allow_api_fallback = getOption("vgi.search_allow_api_fallback", TRUE)
)
```

Arguments

query	Character string. The search query (game title).
limit	Integer. Maximum number of results to return. Defaults to 10.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.
allow_api_fallback	Logical. If TRUE, falls back to the expensive games/game-list API endpoint when no local cache is available. Defaults to TRUE (set options(vgi.search_allow_api_fallback = FALSE) to disable).

Value

A tibble containing search results with game information including steamAppId and name.

Examples

```
## Not run:
# Ensure the VGI_AUTH_TOKEN environment variable is set
# Sys.setenv(VGI_AUTH_TOKEN = "your_auth_token_here")

# Search for games with "valve" in the title
valve_games <- vgi_search_games("valve")
print(valve_games)

# Search with more results
rpg_games <- vgi_search_games("rpg", limit = 50)

## End(Not run)
```

---

vgi_smart_game_search *Smart Game Search using Publisher/Developer Relationships*

Description

This function performs intelligent game searches by finding one game that matches your query, then retrieving ALL games from the same publisher or developer. This is the RIGHT way to search for games - not by searching through a massive cached list.

Usage

```
vgi_smart_game_search(
  query,
  search_type = c("publisher", "developer"),
  limit = 50,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

query	Character string. The search query (partial game name).
search_type	Character. Either "publisher" or "developer" to determine which relationship to follow. Defaults to "publisher".
limit	Integer. Maximum number of results to return. Defaults to 50.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Details

This function implements the CORRECT way to search for games:

1. First, it finds ONE game that matches your query (from cache or API)
2. Then it gets the publisher or developer ID from that game's metadata
3. Finally, it retrieves ALL games from that publisher/developer

This is much smarter than searching through thousands of cached games and actually uses the API's relationship structure properly.

Example: Search for "Battlefield" -> finds one Battlefield game -> gets EA's publisher ID -> returns ALL EA games including all Battlefields.

Value

A tibble containing games from the same publisher/developer with columns:

steamAppId Integer. The Steam App ID

name Character. The game name

publisher Character. Publisher name (if search_type="publisher")

developer Character. Developer name (if search_type="developer")

releaseDate Date. Game release date

Examples

```
## Not run:
# Find all games from Battlefield's publisher (EA)
ea_games <- vgi_smart_game_search("battlefield", search_type = "publisher")
print(ea_games)

# Find all games from the developer of Gothic
gothic_dev_games <- vgi_smart_game_search("gothic", search_type = "developer")
print(gothic_dev_games)

## End(Not run)
```

vgi_steam_market_data *Get Steam Market Data*

Description

Retrieve monthly Steam-wide market totals (units, revenue, releases, users online, accounts) from `/market-data`.

Usage

```
vgi_steam_market_data(
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

`auth_token`	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
`headers`	List. Optional custom headers to include in the API request.

Value

A tibble with one row per month: `period` (`YYYY-MM-01`), `units_total`, `revenue_total`, `releases_total`, `users_online`, `users_ingame`, `accounts_total` and `platform`.

Examples

```
## Not run:
market <- vgi_steam_market_data()
tail(market)

## End(Not run)
```

vgi_top_countries *Get Top Countries for a Game*

Description

Retrieve the countries with the largest share of a Steam game's players. For many games at once use `vgi_all_games_top_countries()`.

Usage

```
vgi_top_countries(
  steam_app_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

steam_app_id	Integer. The Steam App ID of the game.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

A tibble with columns country (ISO code), country_name, percentage and rank, sorted by rank.

Examples

```
## Not run:
vgi_top_countries(steam_app_id = 4019220)

## End(Not run)
```

---

vgi_top_regions *Get Top Regions for a Game*

Description

Convenience wrapper around vgi_insights_player_regions() that returns only the regions tibble.

Usage

```
vgi_top_regions(
  steam_app_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

steam_app_id	Integer. The Steam App ID of the game.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

A tibble with columns region_name, rank and percentage.

Examples

```
## Not run:
vgi_top_regions(steam_app_id = 4019220)

## End(Not run)
```

---

vgi_top_wishlist_countries

Get Top Wishlist Countries for a Game

Description

Retrieve the countries contributing the largest share of a Steam game's wishlists, from v4 /player-insights/games/top-wi
For many games at once use vgi_all_games_wishlist_countries().

Usage

```
vgi_top_wishlist_countries(
  steam_app_id,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

steam_app_id	Integer. The Steam App ID of the game.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Value

A tibble with columns country (ISO code), country_name, percentage and rank, sorted by rank.

Examples

```
## Not run:
vgi_top_wishlist_countries(steam_app_id = 4019220)

## End(Not run)
```

vgi_units_sold_by_date *Get Units Sold Data by Date*

Description

Retrieve units sold data for all games on a specific date, providing a market-wide view of sales performance.

Usage

```
vgi_units_sold_by_date(
  date,
  steam_app_ids = NULL,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

date	Character string or Date. The date for which to retrieve data in "YYYY-MM-DD" format.
steam_app_ids	Numeric vector. Optional. Steam App IDs to filter results. If not provided, returns data for all available games.
auth_token	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
headers	List. Optional custom headers to include in the API request.

Details

Units sold data is crucial for:

- Market share analysis
- Sales velocity tracking
- Launch performance benchmarking
- Seasonal sales pattern identification
- Competitive analysis

The data represents lifetime units sold through the specified date, with daily units calculated from sequential dates.

Value

A data frame with columns:

steamAppId Integer. The Steam App ID

date Character. The date of the data

unitsSold Integer. Cumulative units sold as of this date

dailyUnits Integer. Units sold on this specific day

salesRank Integer. Rank by total units sold

Examples

```
## Not run:
# Get units sold data for a specific date
units_data <- vgi_units_sold_by_date("2024-01-15")

# Top 20 best-selling games
top_sellers <- head(units_data, 20)
cat("Top 20 best-selling games:\n")
print(top_sellers[, c("steamAppId", "unitsSold", "dailyUnits")])

# Calculate previous day's data for daily sales
prev_date <- as.Date("2024-01-15") - 1
units_prev <- vgi_units_sold_by_date(as.character(prev_date))

# Merge to calculate exact daily sales
daily_sales <- merge(units_data, units_prev,
                     by = "steamAppId",
                     suffixes = c("_today", "_yesterday"))
daily_sales$units_sold_today <- daily_sales$unitsSold_today -
                                daily_sales$unitsSold_yesterday

# Find games with highest daily sales
top_daily <- head(daily_sales[order(-daily_sales$units_sold_today), ], 20)
cat("Top 20 games by daily sales:\n")
print(top_daily[, c("steamAppId", "units_sold_today")])

# Analyze sales distribution
hist(log10(units_data$unitsSold + 1),
     breaks = 40,
     main = "Distribution of Total Units Sold (log scale)",
     xlab = "Log10(Units Sold + 1)",
     col = "darkgreen")

# Sales velocity analysis
units_data$sales_per_day <- units_data$unitsSold /
  as.numeric(as.Date("2024-01-15") - as.Date("2020-01-01"))

# Games with sustained high sales velocity
high_velocity <- units_data[units_data$sales_per_day > 100 &
                            units_data$unitsSold > 100000, ]
cat("Games averaging >100 sales/day:", nrow(high_velocity), "\n")
```

```
# Compare with revenue data for average price calculation
revenue_data <- vgi_revenue_by_date("2024-01-15")
pricing <- merge(units_data, revenue_data, by = "steamAppId")
pricing$avg_price <- pricing$revenue / (pricing$unitsSold + 1)

# Find premium-priced successful games
premium_games <- pricing[pricing$avg_price > 40 &
                         pricing$unitsSold > 50000, ]
cat("Premium games (>$40) with >50k sales:", nrow(premium_games), "\n")

## End(Not run)
```

`vgi_wishlists_by_date` *Get Wishlists Data by Date*

Description

Retrieve wishlist counts for all games on a specific date, useful for tracking market-wide interest and anticipation trends.

Usage

```
vgi_wishlists_by_date(
  date,
  steam_app_ids = NULL,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

`date`	Character string or Date. The date for which to retrieve data in "YYYY-MM-DD" format.
`steam_app_ids`	Numeric vector. Optional. Steam App IDs to filter results. If not provided, returns data for all available games.
`auth_token`	Character string. Your VGI API authentication token. Defaults to the VGI_AUTH_TOKEN environment variable.
`headers`	List. Optional custom headers to include in the API request.

Details

This endpoint enables:

- Market-wide wishlist trend analysis
- Identifying rising games before launch
- Tracking pre-release momentum

- Competitive wishlist benchmarking
- Seasonal wishlist pattern analysis

Wishlist data is particularly valuable for unreleased games as it's often the primary engagement metric available.

Value

A data frame with columns:

steamAppId Integer. The Steam App ID

date Character. The date of the data

wishlistCount Integer. Number of wishlists

wishlistRank Integer. Rank by wishlist count

Examples

```
## Not run:
# Get wishlist data for a specific date
wishlists <- vgi_wishlists_by_date("2024-01-15")

# Top 20 most wishlisted games
top_wishlisted <- head(wishlists, 20)
print(top_wishlisted)

# Track weekly wishlist changes
week_ago <- as.Date("2024-01-15") - 7
wishlists_prev <- vgi_wishlists_by_date(as.character(week_ago))

# Calculate weekly growth
growth <- merge(wishlists, wishlists_prev,
                by = "steamAppId",
                suffixes = c("_now", "_prev"))
growth$weekly_change <- growth$wishlistCount_now - growth$wishlistCount_prev
growth$weekly_pct <- (growth$weekly_change / growth$wishlistCount_prev) * 100

# Find fastest growing games
min_wishlists <- 1000  # Only games with substantial wishlists
qualified <- growth[growth$wishlistCount_prev >= min_wishlists, ]
fastest_growing <- head(qualified[order(-qualified$weekly_pct), ], 20)

cat("Fastest growing games (>1000 wishlists):\n")
print(fastest_growing[, c("steamAppId", "wishlistCount_now",
                          "weekly_change", "weekly_pct")])

# Analyze wishlist distribution
hist(log10(wishlists$wishlistCount + 1),
     breaks = 30,
     main = "Distribution of Wishlist Counts (log scale)",
     xlab = "Log10(Wishlist Count + 1)",
     col = "lightgreen")
```

```
# Find games losing wishlists
declining <- growth[growth$weekly_change < -100, ]
cat("Games losing 100+ wishlists this week:", nrow(declining), "\n")

# Seasonal analysis (if you have historical data)
# Compare with same date last year
last_year <- as.Date("2024-01-15") - 365
wishlists_ly <- vgi_wishlists_by_date(as.character(last_year))

yoy <- merge(wishlists, wishlists_ly,
             by = "steamAppId",
             suffixes = c("_now", "_lastyear"))
yoy$yoy_growth <- ((yoy$wishlistCount_now - yoy$wishlistCount_lastyear) /
                   yoy$wishlistCount_lastyear) * 100

# Find games with massive YoY growth
breakout <- yoy[yoy$yoy_growth > 1000 & yoy$wishlistCount_now > 10000, ]
cat("Breakout games (>1000% YoY growth, >10k wishlists):", nrow(breakout), "\n")

## End(Not run)
```

Index

.vgi_cache_dir, 3

apply_rate_limit, 3

create_rate_limiter, 4

get_steam_app_id, 5

print.vgi_yoy_comparison, 5

should_use_rate_limiting, 6

tibble, *10–16*, *22–24*, *26*, *30–32*, *37*, *46–48*, *52, 53*, *55, 56*, *61*, *63–65*

vgi_active_players_by_date, 7
vgi_all_developer_games, 9
vgi_all_developer_games(), *22*
vgi_all_games_player_overlap, 10
vgi_all_games_playtime, 11
vgi_all_games_playtime(), *44*
vgi_all_games_regions, 12
vgi_all_games_top_countries, 13
vgi_all_games_top_countries(), *63*
vgi_all_games_wishlist_countries, 14
vgi_all_games_wishlist_countries(), *65*
vgi_all_publisher_games, 15
vgi_all_publisher_games(), *10*, *53*
vgi_auth_check, 17
vgi_build_cache, 17
vgi_cache_stats, 18
vgi_clear_cache, 18
vgi_concurrent_players_by_date, 19
vgi_developer_games, 22
vgi_developer_games(), *9*
vgi_developer_info, 23
vgi_developer_list, 24
vgi_developers_overview, 21
vgi_developers_overview(), *23*
vgi_fetch_all_games, 25
vgi_fetch_by_ids, 25
vgi_filter_genre, 26
vgi_followers_by_date, 27
vgi_game_list, 30
vgi_game_metadata, 31
vgi_game_metadata_batch, 32
vgi_game_summary, 33
vgi_game_summary_yoy, 34
vgi_games_metadata, 29
vgi_games_metadata(), *31, 32*
vgi_get_games_by_id, 36
vgi_historical_data, 37
vgi_historical_data_by_date, 38
vgi_historical_data_by_date(), *37*
vgi_insights_ccu, 40
vgi_insights_dau_mau, 41
vgi_insights_followers, 42
vgi_insights_player_regions, 43
vgi_insights_player_regions(), *64*
vgi_insights_playtime, 44
vgi_insights_playtime(), *11*
vgi_insights_price_history, 44
vgi_insights_price_history(), *51*
vgi_insights_revenue, 45
vgi_insights_reviews, 46
vgi_insights_units, 47
vgi_insights_wishlists, 48
vgi_load_cache, 49
vgi_peak_ccu_by_ids, 49
vgi_peak_ccu_by_names, 50
vgi_player_overlap, 50
vgi_player_overlap(), *10, 11*
vgi_price_history, 51
vgi_price_history(), *44*
vgi_publisher_games, 53
vgi_publisher_games(), *16*
vgi_publisher_info, 54
vgi_publisher_list, 55
vgi_publishers_overview, 52
vgi_publishers_overview(), *54*

vgi_revenue_by_date, 56
vgi_reviews_by_date, 58
vgi_search_games, 60
vgi_smart_game_search, 61
vgi_steam_market_data, 63
vgi_top_countries, 63
vgi_top_countries(), *13*
vgi_top_regions, 64
vgi_top_regions(), *12*, *43*
vgi_top_wishlist_countries, 65
vgi_top_wishlist_countries(), *14*
vgi_units_sold_by_date, 66
vgi_wishlists_by_date, 68