Package ‘videogameinsightsR’

July 26, 2025

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

Version 0.0.1

Author Phillip Black [aut, cre]

Maintainer Phillip Black <pblack@gameeconomistconsulting.com>

Description Interface to the 'Video Game Insights' API <https://vginsights.com> for video game market analytics and intelligence. Provides functions to retrieve game metadata, developer and publisher information, player statistics (concurrent players, daily/monthly active users), revenue and sales data, review analytics, wishlist tracking, and platform-specific rankings. The package includes data processing utilities to analyze player demographics, track pricing history, calculate player overlap between games, and monitor market trends. Supports analysis across multiple gaming platforms including 'Steam', 'PlayStation', 'Xbox', and 'Nintendo' with unified data structures for cross-platform comparison.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Imports dplyr, httr2, jsonlite, rlang, tibble, tidyr

Suggests ggplot2, gt, knitr, rmarkdown, testthat (>= 3.0.0), withr

Config/testthat/edition 3

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

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

NeedsCompilation no

R topics documented:

`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,
  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.

`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 All Developer Game IDs

Description

Retrieve a comprehensive mapping of all developers to their game IDs, useful for bulk analysis of developer portfolios.

Usage

```
vgi_all_developer_games(
  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.

Details

This endpoint provides a complete developer-to-games mapping, enabling:

- Portfolio size analysis across all developers
- Developer productivity metrics
- Market concentration studies
- Genre specialization analysis
- Developer ranking by output

Note: The gameIds column contains lists, which may need special handling for certain analyses.

Value

A data frame with columns:

developerId Integer. The developer's company ID

gameIds List. A list of Steam App IDs for games by this developer

gameCount Integer. Number of games by this developer

Examples

```
## Not run:
# Get all developer game mappings
dev_games <- vgi_all_developer_games()

# Find most prolific developers
top_devs <- head(dev_games[order(-dev_games$gameCount), ], 20)
cat("Top 20 most prolific developers:\n")
print(top_devs[, c("developerId", "gameCount")])

# Get developer names for context
dev_list <- vgi_developer_list()
top_devs_named <- merge(top_devs, dev_list,
                       by.x = "developerId", by.y = "id")
print(top_devs_named[, c("name", "gameCount")])

# Analyze developer portfolio sizes
hist(dev_games$gameCount[dev_games$gameCount <= 50],
     breaks = 50,
     main = "Distribution of Developer Portfolio Sizes",
     xlab = "Number of Games",
     col = "lightblue")

# Find single-game developers
single_game_devs <- dev_games[dev_games$gameCount == 1, ]
cat("Developers with only one game:", nrow(single_game_devs), "\n")
cat("Percentage of single-game developers:",
    round(nrow(single_game_devs) / nrow(dev_games) * 100, 1), "%\n")

# Analyze specific developer's portfolio
valve_id <- 8  # Example: Valve's ID
valve_games <- dev_games$gameIds[dev_games$developerId == valve_id][[1]]
if (length(valve_games) > 0) {
  cat("Valve has", length(valve_games), "games\n")

  # Get metadata for all Valve games
  valve_metadata <- vgi_game_metadata_batch(valve_games)
  print(valve_metadata[, c("name", "releaseDate")])
}

# Find developers with similar portfolio sizes
target_size <- 10
similar_devs <- dev_games[dev_games$gameCount >= target_size - 2 &
                         dev_games$gameCount <= target_size + 2, ]
cat("Developers with 8-12 games:", nrow(similar_devs), "\n")

# Calculate total games in database
total_games <- sum(dev_games$gameCount)
unique_games <- length(unique(unlist(dev_games$gameIds)))
cat("Total developer-game relationships:", total_games, "\n")
cat("Unique games:", unique_games, "\n")
cat("Average developers per game:", round(total_games / unique_games, 2), "\n")

## End(Not run)
```

`vgi_all_games_metadata`

Get Metadata for All Games

Description

Retrieve comprehensive metadata for all games in the database, providing essential information for game identification and categorization.

Usage

```
vgi_all_games_metadata(
  limit = 1000,
  offset = 0,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

`limit`	Integer. Maximum number of games to return (default 1000). Use NULL to return all games (may be very large).
`offset`	Integer. Number of games to skip for pagination (default 0).
`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 provides the foundation for:

- Building game catalogs and databases
- Genre and tag analysis
- Release pattern studies
- Price point analysis
- Developer/publisher relationships

Note: This endpoint may return a very large dataset. Consider using pagination or caching the results for repeated use.

Value

A data frame with columns:

steamAppId Integer. The Steam App ID

name Character. Game title

releaseDate Character. Release date

developer Character. Primary developer name

publisher Character. Primary publisher name

genres List. Game genres

tags List. Steam tags

price Numeric. Current price in USD

description Character. Game description

Examples

```
## Not run:
# Get first 1000 games
games_metadata <- vgi_all_games_metadata(limit = 1000)

# Basic statistics
cat("Total games:", nrow(games_metadata), "\n")
cat("Date range:", min(games_metadata$releaseDate), "to",
    max(games_metadata$releaseDate), "\n")

# Price analysis
price_stats <- summary(games_metadata$price)
print(price_stats)

# Free vs paid games
free_games <- sum(games_metadata$price == 0)
cat("Free games:", free_games,
    "(", round(free_games/nrow(games_metadata)*100, 1), "%)\n")

# Genre analysis
all_genres <- unlist(games_metadata$genres)
genre_counts <- sort(table(all_genres), decreasing = TRUE)
cat("Top 10 genres:\n")
print(head(genre_counts, 10))

# Tag analysis for trends
all_tags <- unlist(games_metadata$tags)
tag_counts <- sort(table(all_tags), decreasing = TRUE)
cat("Top 20 tags:\n")
print(head(tag_counts, 20))

# Release patterns by year
games_metadata$year <- format(as.Date(games_metadata$releaseDate), "%Y")
yearly_releases <- table(games_metadata$year)

barplot(yearly_releases[names(yearly_releases) >= "2015"],
        main = "Games Released per Year",
        xlab = "Year",
        ylab = "Number of Games",
        las = 2,
        col = "steelblue")

# Developer analysis
dev_counts <- sort(table(games_metadata$developer), decreasing = TRUE)
cat("Most prolific developers:\n")
print(head(dev_counts, 10))

# Price tier analysis
games_metadata$price_tier <- cut(games_metadata$price,
                                 breaks = c(-0.01, 0, 9.99, 19.99,
                                         39.99, 59.99, Inf),
                                 labels = c("Free", "<$10", "$10-20",
                                          "$20-40", "$40-60", ">$60"))

tier_dist <- table(games_metadata$price_tier)
pie(tier_dist,
    main = "Games by Price Tier",
    col = rainbow(length(tier_dist)))

# Find games by specific criteria
# Recent indie games
recent_indie <- games_metadata[
  games_metadata$year >= "2023" &
  sapply(games_metadata$tags, function(t) "Indie" %in% t) &
  games_metadata$price < 30,
]
cat("Recent indie games under $30:", nrow(recent_indie), "\n")

# Export for external analysis
# write.csv(games_metadata, "all_games_metadata.csv", row.names = FALSE)

# Paginated retrieval for large datasets
# all_games <- list()
# offset <- 0
# repeat {
#   batch <- vgi_all_games_metadata(limit = 1000, offset = offset)
#   if (nrow(batch) == 0) break
#   all_games[[length(all_games) + 1]] <- batch
#   offset <- offset + 1000
#   cat("Retrieved", offset, "games...\n")
# }
# all_games_df <- do.call(rbind, all_games)

## End(Not run)
```

vgi_all_games_player_overlap

Get Player Overlap Data for All Games

Description

Retrieve player overlap statistics for all games, showing which games share the most players and identifying gaming ecosystem connections.

Usage

```
vgi_all_games_player_overlap(
  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.

Details

Player overlap data reveals:

- Direct competitors (high overlap = similar audience)
- Complementary games (moderate overlap = cross-promotion opportunities)
- Genre clusters and gaming ecosystems
- Sequel/franchise connections
- Platform-specific communities

The topOverlaps list column contains detailed overlap data that can be analyzed for deeper insights.

Value

A data frame with columns:

steamAppId Integer. The Steam App ID

topOverlaps List. Top overlapping games with overlap metrics

overlapCount Integer. Number of games with significant overlap

topOverlapGame Integer. Steam App ID of most overlapping game

topOverlapPct Numeric. Percentage overlap with top game

Examples

```
## Not run:
# Get player overlap data for all games
overlap_data <- vgi_all_games_player_overlap()

# Find games with highest overlap percentages
high_overlap <- overlap_data[overlap_data$topOverlapPct > 50, ]
cat("Games with >50% overlap with another game:", nrow(high_overlap), "\n")

# These are likely sequels, expansions, or very similar games
print(head(high_overlap[, c("steamAppId", "topOverlapGame", "topOverlapPct")]))

# Analyze overlap patterns
hist(overlap_data$topOverlapPct,
     breaks = 50,
     main = "Distribution of Maximum Player Overlap",
     xlab = "Top Overlap Percentage",
     col = "lightcoral")

# Find gaming clusters (mutual high overlap)
# Check if game A's top overlap is game B, and vice versa
mutual_overlaps <- overlap_data[
  mapply(function(id, top_id) {
    if (is.na(top_id)) return(FALSE)
    overlap_data$topOverlapGame[overlap_data$steamAppId == top_id] == id
  }, overlap_data$steamAppId, overlap_data$topOverlapGame),
]
cat("Games with mutual top overlap:", nrow(mutual_overlaps), "\n")

# Extract detailed overlap data for analysis
# Find games that overlap with a specific game
target_game <- 730  # Example: Counter-Strike 2
games_overlapping_target <- overlap_data[
  sapply(overlap_data$topOverlaps, function(overlaps) {
    if (is.null(overlaps)) return(FALSE)
    target_game %in% overlaps$steamAppId
  }),
]
cat("Games with significant overlap with game", target_game, ":",
    nrow(games_overlapping_target), "\n")

# Build a gaming ecosystem map
# Extract all overlap relationships
all_overlaps <- do.call(rbind, lapply(seq_len(nrow(overlap_data)), function(i) {
  game_id <- overlap_data$steamAppId[i]
  overlaps <- overlap_data$topOverlaps[[i]]
  if (is.null(overlaps) || nrow(overlaps) == 0) return(NULL)

  data.frame(
    from = game_id,
    to = overlaps$steamAppId[1:min(5, nrow(overlaps))],
    overlap_pct = overlaps$overlapPercentage[1:min(5, nrow(overlaps))]
  )
}))

# Find most connected games (hubs in the network)
connection_counts <- table(c(all_overlaps$from, all_overlaps$to))
hubs <- head(sort(connection_counts, decreasing = TRUE), 20)
cat("Most connected games (appear in many overlaps):\n")
print(hubs)

# Genre affinity analysis (would need genre data)
# Games with high overlap likely share genres
# Could cluster games based on overlap patterns

# Find isolated games (low overlap with any other game)
isolated_games <- overlap_data[overlap_data$topOverlapPct < 5 |
                               is.na(overlap_data$topOverlapPct), ]
cat("Games with <5% overlap (unique/niche):", nrow(isolated_games), "\n")

## End(Not run)
```

`vgi_all_games_playtime` *Get Playtime Data for All Games*

Description

Retrieve playtime statistics for all games in the database, providing a comprehensive view of player engagement across the market.

Usage

```
vgi_all_games_playtime(
  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.

Details

Playtime data is crucial for understanding:

- Game engagement and stickiness
- Content depth and replayability
- Player satisfaction (high playtime often = high satisfaction)
- Genre-specific engagement patterns
- Value proposition (playtime per dollar)

Average playtime can be skewed by dedicated players, while median provides a better sense of typical player engagement.

Value

A data frame with columns:

steamAppId Integer. The Steam App ID

avgPlaytime Numeric. Average playtime in hours

medianPlaytime Numeric. Median playtime in hours

totalPlaytime Numeric. Total playtime across all players in hours

playtimeRank Integer. Rank by average playtime

Examples

```
## Not run:
# Get playtime data for all games
playtime_data <- vgi_all_games_playtime()

# Top 20 most played games by average playtime
top_played <- head(playtime_data, 20)
cat("Top 20 games by average playtime:\n")
print(top_played[, c("steamAppId", "avgPlaytime", "medianPlaytime")])

# Find games with high engagement
high_engagement <- playtime_data[playtime_data$avgPlaytime > 100, ]
cat("Games with >100 hours average playtime:", nrow(high_engagement), "\n")

# Analyze playtime distribution
hist(log10(playtime_data$avgPlaytime + 1),
     breaks = 40,
     main = "Distribution of Average Playtime (log scale)",
     xlab = "Log10(Avg Playtime + 1)",
     col = "orange")

# Compare average vs median to find games with dedicated players
playtime_data$avg_median_ratio <- playtime_data$avgPlaytime /
                                  (playtime_data$medianPlaytime + 0.1)

# Games where average is much higher than median (cult followings)
cult_games <- playtime_data[playtime_data$avg_median_ratio > 5 &
                           playtime_data$avgPlaytime > 20, ]
cat("Games with cult followings (avg >> median):", nrow(cult_games), "\n")

# Combine with revenue data for value analysis
revenue_data <- vgi_revenue_by_date(Sys.Date() - 1)
value_analysis <- merge(playtime_data, revenue_data, by = "steamAppId")

# Calculate hours per dollar (value metric)
units_data <- vgi_units_sold_by_date(Sys.Date() - 1)
value_analysis <- merge(value_analysis, units_data, by = "steamAppId")
value_analysis$avg_price <- value_analysis$revenue /
                           (value_analysis$unitsSold + 1)
value_analysis$hours_per_dollar <- value_analysis$avgPlaytime /
                                   (value_analysis$avg_price + 0.01)

# Best value games (high playtime, reasonable price)
best_value <- value_analysis[value_analysis$hours_per_dollar > 2 &
                            value_analysis$avg_price < 60 &
                            value_analysis$unitsSold > 10000, ]
best_value <- head(best_value[order(-best_value$hours_per_dollar), ], 20)
cat("Best value games (hours per dollar):\n")
print(best_value[, c("steamAppId", "avgPlaytime", "avg_price",
                    "hours_per_dollar")])

# Genre analysis (would need genre data)
# Multiplayer games typically have higher playtime
likely_multiplayer <- playtime_data[playtime_data$avgPlaytime > 50 &
                                   playtime_data$medianPlaytime > 20, ]
cat("Likely multiplayer/service games:", nrow(likely_multiplayer), "\n")

## End(Not run)
```

---

vgi_all_games_regions *Get Regional Distribution Data for All Games*

Description

Retrieve regional player distribution for all games, showing how players are distributed across major world regions.

Usage

```
vgi_all_games_regions(
  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.

Details

Regional data provides high-level insights for:

- Global market strategy
- Server infrastructure planning
- Marketing budget allocation
- Content scheduling (time zones)
- Localization priorities

Regions are aggregated from individual country data using standard geographic classifications.

Value

A data frame with columns:

steamAppId Integer. The Steam App ID

northAmerica Numeric. Percentage of players from North America

europe Numeric. Percentage of players from Europe

asia Numeric. Percentage of players from Asia

southAmerica Numeric. Percentage of players from South America

oceania Numeric. Percentage of players from Oceania

africa Numeric. Percentage of players from Africa

middleEast Numeric. Percentage of players from Middle East

dominantRegion Character. Region with highest player percentage

Examples

```
## Not run:
# Get regional data for all games
regions_data <- vgi_all_games_regions()

# Find games by dominant region
dominant_regions <- table(regions_data$dominantRegion)
print(dominant_regions)

# Visualize regional distribution
pie(dominant_regions,
    main = "Games by Dominant Region",
    col = rainbow(length(dominant_regions)))

# Find globally balanced games
```

```
regions_data$max_region_pct <- pmax(regions_data$northAmerica,
                                    regions_data$europe,
                                    regions_data$asia,
                                    regions_data$southAmerica,
                                    regions_data$oceania,
                                    regions_data$africa,
                                    regions_data$middleEast)

balanced_games <- regions_data[regions_data$max_region_pct < 40, ]
cat("Games with no region >40%:", nrow(balanced_games), "\n")

# Compare Western vs Eastern games
regions_data$western_pct <- regions_data$northAmerica +
                           regions_data$europe +
                           regions_data$oceania
regions_data$eastern_pct <- regions_data$asia +
                           regions_data$middleEast

western_games <- regions_data[regions_data$western_pct > 70, ]
eastern_games <- regions_data[regions_data$eastern_pct > 70, ]

cat("Western-dominated games (>70%):", nrow(western_games), "\n")
cat("Eastern-dominated games (>70%):", nrow(eastern_games), "\n")

# Analyze emerging markets
emerging_markets <- regions_data$southAmerica +
                   regions_data$africa +
                   regions_data$middleEast

emerging_focused <- regions_data[emerging_markets > 30, ]
cat("Games with >30% from emerging markets:", nrow(emerging_focused), "\n")

# Create regional profile heatmap (requires additional packages)
# library(ggplot2)
# library(reshape2)
#
# top_100 <- head(regions_data, 100)
# regions_matrix <- top_100[, c("steamAppId", "northAmerica", "europe",
#                              "asia", "southAmerica", "oceania",
#                              "africa", "middleEast")]
# regions_long <- melt(regions_matrix, id.vars = "steamAppId")
#
# ggplot(regions_long, aes(x = variable, y = steamAppId, fill = value)) +
#   geom_tile() +
#   scale_fill_gradient(low = "white", high = "darkblue") +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
#   labs(title = "Regional Distribution Heatmap (Top 100 Games)",
#        x = "Region", y = "Game", fill = "Player %")

# Find region-specific genres (would need genre data)
# Asia-focused games might be more likely to be MMOs or mobile ports
asia_focused <- regions_data[regions_data$asia > 50, ]
cat("Asia-focused games (>50% Asian players):", nrow(asia_focused), "\n")

## End(Not run)
```

vgi_all_games_top_countries

Get Top Countries Data for All Games

Description

Retrieve top countries by player count for all games, providing insights into global gaming preferences and regional market dynamics.

Usage

```
vgi_all_games_top_countries(
  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.

Details

This endpoint helps identify:

- Games with global vs regional appeal
- Regional gaming preferences
- Localization opportunities
- Market penetration patterns
- Cultural gaming trends

The topCountries list column contains detailed country breakdowns that can be expanded for deeper analysis.

Value

A data frame with columns:

steamAppId Integer. The Steam App ID

topCountries List. Top countries with their player percentages

countryCount Integer. Number of countries in the data

topCountry Character. The #1 country by player count

topCountryPct Numeric. Percentage of players from top country

Examples

```
## Not run:
# Get top countries for all games
countries_data <- vgi_all_games_top_countries()

# Find games dominated by specific countries
us_dominated <- countries_data[countries_data$topCountry == "US" &
                               countries_data$topCountryPct > 50, ]
cat("Games where >50% of players are from US:", nrow(us_dominated), "\n")

# Find globally diverse games
global_games <- countries_data[countries_data$topCountryPct < 20 &
                               countries_data$countryCount > 50, ]
cat("Globally diverse games (<20% from any country):", nrow(global_games), "\n")

# Analyze regional preferences
top_countries_summary <- table(countries_data$topCountry)
top_10_countries <- head(sort(top_countries_summary, decreasing = TRUE), 10)

barplot(top_10_countries,
        main = "Countries Most Often #1 in Games",
        xlab = "Country",
        ylab = "Number of Games Where #1",
        las = 2,
        col = "steelblue")

# Extract detailed country data for a specific game
game_id <- 730  # Example game
game_countries <- countries_data$topCountries[
  countries_data$steamAppId == game_id][[1]]
if (!is.null(game_countries)) {
  print(head(game_countries, 10))
}

# Find games popular in specific regions
# Extract games where China is in top 3
china_popular <- countries_data[sapply(countries_data$topCountries,
  function(tc) {
    if (is.null(tc) || nrow(tc) < 3) return(FALSE)
    "CN" %in% tc$country[1:3]
  }), ]
cat("Games where China is in top 3 countries:", nrow(china_popular), "\n")

# Calculate market concentration
countries_data$top3_concentration <- sapply(countries_data$topCountries,
  function(tc) {
    if (is.null(tc) || nrow(tc) < 3) return(NA)
    sum(tc$percentage[1:3])
  })

# Games with highest geographic concentration
concentrated <- countries_data[!is.na(countries_data$top3_concentration) &
                               countries_data$top3_concentration > 70, ]
cat("Games where top 3 countries >70% of players:", nrow(concentrated), "\n")

# Regional gaming hours analysis
# Games popular in Asia vs Americas vs Europe
asia_countries <- c("CN", "JP", "KR", "TW", "HK", "SG", "TH", "ID")
americas_countries <- c("US", "CA", "BR", "MX", "AR", "CL", "CO")
europe_countries <- c("DE", "FR", "GB", "IT", "ES", "PL", "NL", "SE")

countries_data$asia_pct <- sapply(countries_data$topCountries,
  function(tc) {
    if (is.null(tc)) return(0)
    sum(tc$percentage[tc$country %in% asia_countries])
  })

asia_focused <- countries_data[countries_data$asia_pct > 50, ]
cat("Games with >50% Asian players:", nrow(asia_focused), "\n")

## End(Not run)
```

vgi_all_games_wishlist_countries

Get Top Wishlist Countries Data for All Games

Description

Retrieve top countries by wishlist count for all games, providing insights into global interest patterns and pre-release market potential.

Usage

```
vgi_all_games_wishlist_countries(
  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.

Details

Wishlist geographic data reveals:

- Pre-launch interest by region
- Marketing effectiveness across countries
- Localization priorities
- Launch strategy optimization
- Conversion potential by region

Comparing wishlist distribution with actual player distribution can reveal untapped markets or conversion challenges.

Value

A data frame with columns:

steamAppId Integer. The Steam App ID

topWishlistCountries List. Top countries with wishlist percentages

wishlistCountryCount Integer. Number of countries with wishlists

topWishlistCountry Character. The #1 country by wishlist count

topWishlistCountryPct Numeric. Percentage of wishlists from top country

Examples

```
## Not run:
# Get wishlist country data for all games
wishlist_countries <- vgi_all_games_wishlist_countries()

# Compare with player country data
player_countries <- vgi_all_games_top_countries()

# Merge to analyze wishlist vs player patterns
geo_comparison <- merge(wishlist_countries, player_countries,
                       by = "steamAppId",
                       suffixes = c("_wishlist", "_player"))

# Find games where wishlist and player geography differ
geo_comparison$country_match <-
  geo_comparison$topWishlistCountry == geo_comparison$topCountry

mismatched <- geo_comparison[!geo_comparison$country_match, ]
cat("Games where top wishlist country != top player country:",
    nrow(mismatched), "\n")

# Analyze wishlist concentration
hist(wishlist_countries$topWishlistCountryPct,
     breaks = 30,
     main = "Wishlist Geographic Concentration",
     xlab = "Top Country Wishlist %",
     col = "darkgreen")

# Find games with global wishlist appeal
global_wishlist <- wishlist_countries[
  wishlist_countries$topWishlistCountryPct < 25 &
  wishlist_countries$wishlistCountryCount > 50, ]
cat("Games with global wishlist distribution:", nrow(global_wishlist), "\n")

# Regional wishlist patterns
wishlist_top_countries <- table(wishlist_countries$topWishlistCountry)
player_top_countries <- table(player_countries$topCountry)

# Compare which countries dominate wishlists vs players
country_comparison <- merge(
  data.frame(country = names(wishlist_top_countries),
             wishlist_games = as.numeric(wishlist_top_countries)),
  data.frame(country = names(player_top_countries),
             player_games = as.numeric(player_top_countries)),
  by = "country", all = TRUE
)
country_comparison[is.na(country_comparison)] <- 0
country_comparison$wishlist_player_ratio <-
  country_comparison$wishlist_games / (country_comparison$player_games + 1)

# Countries that wishlist more than they play
high_wishlist_countries <- country_comparison[
  country_comparison$wishlist_player_ratio > 1.5 &
  country_comparison$wishlist_games > 10, ]
cat("Countries that wishlist disproportionately:\n")
print(high_wishlist_countries[order(-high_wishlist_countries$wishlist_player_ratio), ])

# Extract detailed data for emerging markets
emerging_markets <- c("BR", "IN", "MX", "TR", "PL")

emerging_wishlist_share <- sapply(wishlist_countries$topWishlistCountries,
  function(countries) {
    if (is.null(countries)) return(0)
    sum(countries$percentage[countries$country %in% emerging_markets])
  })

high_emerging <- wishlist_countries[emerging_wishlist_share > 30, ]
cat("Games with >30% wishlists from emerging markets:", nrow(high_emerging), "\n")

# Wishlist velocity by region (would need time series data)
# Could identify which regions are gaining momentum

## End(Not run)
```

vgi_all_publisher_games

Get All Publisher Game IDs

Description

Retrieve a comprehensive mapping of all publishers to their game IDs, useful for analyzing publishing portfolios and market dynamics.

Usage

```
vgi_all_publisher_games(
  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.

Details

This endpoint provides a complete publisher-to-games mapping, enabling:

- Publishing portfolio analysis
- Market share calculations
- Publisher strategy assessment
- Competitive landscape mapping
- Publisher ranking by catalog size

Publishers often have larger portfolios than developers as they may publish games from multiple development studios.

Value

A data frame with columns:

publisherId Integer. The publisher's company ID

gameIds List. A list of Steam App IDs for games by this publisher

gameCount Integer. Number of games by this publisher

Examples

```
## Not run:
# Get all publisher game mappings
pub_games <- vgi_all_publisher_games()

# Find largest publishers by catalog size
top_pubs <- head(pub_games[order(-pub_games$gameCount), ], 20)
cat("Top 20 largest publishers:\n")
print(top_pubs[, c("publisherId", "gameCount")])

# Get publisher names for context
pub_list <- vgi_publisher_list()
top_pubs_named <- merge(top_pubs, pub_list,
                        by.x = "publisherId", by.y = "id")
print(top_pubs_named[, c("name", "gameCount")])

# Compare with developer portfolios
dev_games <- vgi_all_developer_games()
cat("Average games per developer:", round(mean(dev_games$gameCount), 1), "\n")
cat("Average games per publisher:", round(mean(pub_games$gameCount), 1), "\n")

# Find mega-publishers (>100 games)
mega_pubs <- pub_games[pub_games$gameCount > 100, ]
cat("Publishers with >100 games:", nrow(mega_pubs), "\n")

# Analyze publisher concentration
total_pub_games <- sum(pub_games$gameCount)
top_10_pub_games <- sum(head(pub_games$gameCount, 10))
concentration <- (top_10_pub_games / total_pub_games) * 100
cat(sprintf("Top 10 publishers control %.1f%% of published games\n",
            concentration))

# Find publishers that also develop
```

```
dev_list <- vgi_developer_list()
pub_dev_overlap <- intersect(pub_list$name, dev_list$name)
cat("Companies that both publish and develop:",
    length(pub_dev_overlap), "\n")

# Analyze specific publisher's portfolio
ea_id <- 1  # Example: Electronic Arts
ea_games <- pub_games$gameIds[pub_games$publisherId == ea_id][[1]]
if (length(ea_games) > 0) {
  cat("EA has published", length(ea_games), "games\n")

  # Get recent EA releases
  ea_metadata <- vgi_game_metadata_batch(ea_games)
  recent_ea <- ea_metadata[as.Date(ea_metadata$releaseDate) >
                          as.Date("2023-01-01"), ]
  cat("EA games released since 2023:", nrow(recent_ea), "\n")
}

# Distribution of publisher sizes
pub_size_dist <- table(cut(pub_games$gameCount,
                          breaks = c(1, 2, 5, 10, 20, 50, 100, Inf),
                          labels = c("1", "2-4", "5-9", "10-19",
                                     "20-49", "50-99", "100+"),
                          right = FALSE))

barplot(pub_size_dist,
        main = "Distribution of Publishers by Portfolio Size",
        xlab = "Number of Games Published",
        ylab = "Number of Publishers",
        col = "purple")

## End(Not run)
```

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,
  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.

`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_developer_games *Get Game IDs by Developer*

Description

Retrieve a list of Steam App IDs for all games by a specific developer.

Usage

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

Arguments

`company_id`	Integer. The developer's company ID.
`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 now returns only game IDs instead of full game metadata. To get detailed information about each game, use `vgi_game_metadata()` with the returned IDs.

This approach is more efficient when you only need to:

- Count the number of games by a developer
- Check if a developer made a specific game
- Get a subset of games for detailed analysis

Value

A numeric vector of Steam App IDs for games developed by this company.

Examples

```
## Not run:
# Get all game IDs for a developer
game_ids <- vgi_developer_games(company_id = 24569)

# Count games by this developer
cat("Total games:", length(game_ids), "\n")

# Get detailed info for the first 5 games
if (length(game_ids) > 0) {
  first_five <- head(game_ids, 5)
  game_details <- lapply(first_five, vgi_game_metadata)

  # Display game names
  for (game in game_details) {
    cat(game$name, "(", game$steamAppId, ")\n")
  }
}

# Check if developer made a specific game
target_game_id <- 730
if (target_game_id %in% game_ids) {
  cat("Yes, this developer made game", target_game_id, "\n")
}

# Analyze developer's recent releases
if (length(game_ids) > 0) {
  # Get metadata for all games
  all_games <- vgi_game_metadata_batch(game_ids)

  # Find games released in last 2 years
  recent_games <- all_games[
    as.Date(all_games$releaseDate) > Sys.Date() - 730,
  ]

  cat("Games released in last 2 years:", nrow(recent_games), "\n")
}

## End(Not run)
```

vgi_developer_info *Get Developer Information*

Description

Retrieve detailed information about a game developer.

Usage

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

Arguments

`company_id` Integer. The developer's company ID.

`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

Developer information can be used to:

- Research company backgrounds and history
- Analyze developer portfolio performance
- Identify experienced developers in specific genres
- Track developer growth over time

Value

A list containing developer information with fields:

companyId Integer. The company ID

name Character. Developer name

foundedDate Character. Date the company was founded

headquarters Character. Company headquarters location

employeeCount Integer. Number of employees

website Character. Company website URL

description Character. Company description

totalGames Integer. Total number of games developed

activeGames Integer. Number of currently active games

Examples

```
## Not run:
# Get information about a developer
dev_info <- vgi_developer_info(company_id = 24569)

# Display developer details
cat("Developer:", dev_info$name, "\n")
cat("Founded:", dev_info$foundedDate, "\n")
cat("Total Games:", dev_info$totalGames, "\n")
cat("Active Games:", dev_info$activeGames, "\n")

# Check developer size
if (!is.null(dev_info$employeeCount)) {
  if (dev_info$employeeCount < 10) {
    cat("This is an indie developer\n")
  } else if (dev_info$employeeCount < 100) {
    cat("This is a mid-sized developer\n")
  } else {
    cat("This is a large developer\n")
  }
}

## End(Not run)
```

`vgi_developer_list` *Get Complete Developer List*

Description

Retrieve a list of all game developers in the Video Game Insights database.

Usage

```
vgi_developer_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.

Details

This endpoint is useful for:

- Discovering all tracked developers
- Building developer selection interfaces
- Finding developer IDs for further queries
- Analyzing the developer ecosystem

Value

A data frame with columns:

id Integer. The company ID

name Character. The developer name

Examples

```
## Not run:
# Get all developers
all_devs <- vgi_developer_list()
cat("Total developers:", nrow(all_devs), "\n")

# Search for developers by name
valve_devs <- all_devs[grep("Valve", all_devs$name, ignore.case = TRUE), ]
print(valve_devs)

# Find developer ID by exact name
dev_id <- all_devs$id[all_devs$name == "Valve Corporation"]
if (length(dev_id) > 0) {
  cat("Valve Corporation ID:", dev_id, "\n")

  # Get more info about this developer
  valve_info <- vgi_developer_info(dev_id)
  print(valve_info)
}

# Analyze developer names
# Find developers with "Studios" in name
studios <- all_devs[grep("Studios", all_devs$name), ]
cat("Developers with 'Studios':", nrow(studios), "\n")

# Find indie developers (often individual names or small teams)
short_names <- all_devs[nchar(all_devs$name) < 15, ]
cat("Developers with short names:", nrow(short_names), "\n")

## 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,
  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.
`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_game_list *Get Complete Game List*

Description

Retrieve a list of all games available in the Video Game Insights database. This is a lightweight endpoint that returns game IDs and names only.

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.

Details

This endpoint is useful for:

- Getting a complete inventory of tracked games
- Building game selection interfaces
- Caching game names for ID lookups
- Validating game IDs before making other API calls

Note: This endpoint may return thousands of games. Consider caching the results locally to avoid repeated API calls.

Value

A data frame with columns:

id Integer. The Steam App ID

name Character. The game name

Examples

```
## Not run:
# Get all games
all_games <- vgi_game_list()
cat("Total games in database:", nrow(all_games), "\n")

# Search for games by name
cs_games <- all_games[grep("Counter-Strike", all_games$name, ignore.case = TRUE), ]
print(cs_games)

# Find game ID by exact name
game_id <- all_games$id[all_games$name == "Counter-Strike 2"]
if (length(game_id) > 0) {
  cat("Counter-Strike 2 ID:", game_id, "\n")
}

# Get random sample of games for analysis
sample_games <- all_games[sample(nrow(all_games), 10), ]
print(sample_games)

# Cache results for future use
saveRDS(all_games, "vgi_game_list_cache.rds")
# Later: all_games <- readRDS("vgi_game_list_cache.rds")

## End(Not run)
```

vgi_game_metadata *Get Game Metadata from Video Game Insights*

Description

Retrieves detailed metadata for a specific game using its Steam App ID.

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 tibble containing game metadata including name, release date, price, genres, categories, developers, publishers, and more.

Examples

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

# Get metadata for Valheim (Steam App ID: 892970)
valheim_data <- vgi_game_metadata(892970)
print(valheim_data)

## End(Not run)
```

`vgi_game_metadata_batch` *Get Batch Game Metadata from Video Game Insights*

Description

Retrieves metadata for multiple games using their Steam App IDs. Since the new API doesn't have a batch endpoint, this function makes multiple individual requests and combines the results.

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 containing metadata for all requested games. Each row represents one game with columns for name, release date, price, genres, categories, developers, publishers, and more.

Examples

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

# Get metadata for multiple games
game_ids <- c(892970, 1245620, 105600) # Valheim, Elden Ring, Terraria
games_data <- vgi_game_metadata_batch(game_ids)
print(games_data)

## End(Not run)
```

`vgi_game_rankings` *Get Game Rankings*

Description

Retrieve rankings for all games across various metrics including reviews, revenue, units sold, followers, and playtime.

Usage

```
vgi_game_rankings(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.

Details

Rankings provide a comprehensive view of game performance across metrics:

- Lower rank numbers indicate better performance (1 = best)
- Percentiles show the percentage of games that rank below
- Use multiple metrics to get a balanced view of game success
- Recent sales rankings help identify trending games

Value

A data frame containing rankings for each game with columns:

steamAppId Integer. The Steam App ID

positiveReviewsRank Integer. Rank by positive reviews

positiveReviewsPrct Numeric. Percentile for positive reviews

totalRevenueRank Integer. Rank by total revenue

totalRevenuePrct Numeric. Percentile for total revenue

totalUnitsSoldRank Integer. Rank by total units sold

totalUnitsSoldPrct Numeric. Percentile for total units sold

yesterdayUnitsSoldRank Integer. Rank by yesterday's units sold

yesterdayUnitsSoldPrct Numeric. Percentile for yesterday's units sold

followersRank Integer. Rank by follower count

followersPrct Numeric. Percentile for followers

avgPlaytimeRank Integer. Rank by average playtime

avgPlaytimePrct Numeric. Percentile for average playtime

Examples

```
## Not run:
# Get all game rankings
rankings <- vgi_game_rankings()

# Find top 10 games by revenue
top_revenue <- head(rankings[order(rankings$totalRevenueRank), ], 10)
print(top_revenue[, c("steamAppId", "totalRevenueRank", "totalRevenuePrct")])

# Find games that rank well across multiple metrics
# (top 100 in both revenue and reviews)
top_overall <- rankings[
  rankings$totalRevenueRank <= 100 &
  rankings$positiveReviewsRank <= 100,
]
print(paste("Games in top 100 for both revenue and reviews:", nrow(top_overall)))

# Identify trending games (high recent sales relative to total)
rankings$trending_score <- rankings$totalUnitsSoldRank / rankings$yesterdayUnitsSoldRank
trending <- head(rankings[order(rankings$trending_score, decreasing = TRUE), ], 20)

# Create a scatter plot of revenue vs reviews rankings
plot(rankings$totalRevenueRank, rankings$positiveReviewsRank,
     pch = 19, col = rgb(0, 0, 1, 0.1),
     xlab = "Revenue Rank", ylab = "Reviews Rank",
     main = "Game Rankings: Revenue vs Reviews")
abline(a = 0, b = 1, col = "red", lty = 2)

# Find hidden gems (great reviews but lower revenue)
hidden_gems <- rankings[
  rankings$positiveReviewsRank <= 50 &
  rankings$totalRevenueRank > 200,
]
```

```
print(paste("Hidden gems found:", nrow(hidden_gems)))

## End(Not run)
```

vgi_historical_data *Get Historical Game Data*

Description

Retrieve comprehensive historical data for a specific game including all available metrics over time.

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

This endpoint provides a comprehensive historical view of a game's performance across all tracked metrics. This is useful for:

- Creating detailed performance dashboards
- Analyzing long-term trends
- Correlating different metrics (e.g., price changes vs. sales)
- Building predictive models
- Generating comprehensive reports

The data typically spans from the game's release date to the present, with different metrics having different update frequencies.

Value

A list containing historical data with components:

steamAppId Integer. The Steam App ID

revenue Data frame with date and revenue columns

unitsSold Data frame with date and units sold columns

concurrentPlayers Data frame with date and concurrent players columns

activePlayers Data frame with date, DAU, and MAU columns

reviews Data frame with date, positive, and negative review counts

wishlists Data frame with date and wishlist count columns

followers Data frame with date and follower count columns

priceHistory Data frame with date, currency, and price columns

Examples

```
## Not run:
# Get all historical data for a game
historical <- vgi_historical_data(steam_app_id = 730)

# Plot revenue over time
if (!is.null(historical$revenue)) {
  plot(as.Date(historical$revenue$date), historical$revenue$revenue,
       type = "l", col = "green", lwd = 2,
       xlab = "Date", ylab = "Revenue ($)",
       main = "Revenue Over Time")
}

# Analyze review sentiment over time
if (!is.null(historical$reviews)) {
  historical$reviews$positiveRatio <- historical$reviews$positive /
    (historical$reviews$positive + historical$reviews$negative)

  plot(as.Date(historical$reviews$date), historical$reviews$positiveRatio,
       type = "l", col = "blue", lwd = 2,
       xlab = "Date", ylab = "Positive Review Ratio",
       main = "Review Sentiment Over Time")
  abline(h = 0.7, col = "green", lty = 2)
  abline(h = 0.5, col = "orange", lty = 2)
}

# Correlate price changes with sales
if (!is.null(historical$priceHistory) && !is.null(historical$unitsSold)) {
  # Find USD prices
  usd_prices <- historical$priceHistory[historical$priceHistory$currency == "USD", ]

  # Match dates between price and units sold
  matched_data <- merge(usd_prices, historical$unitsSold,
                       by = "date", all = FALSE)

  if (nrow(matched_data) > 0) {
    plot(matched_data$price, matched_data$unitsSold,
         pch = 19, col = "darkblue",
         xlab = "Price (USD)", ylab = "Units Sold",
         main = "Price vs. Sales Correlation")
  }
}

# Calculate growth metrics
if (!is.null(historical$followers)) {
  n <- nrow(historical$followers)
  if (n > 30) {
    growth_30d <- (historical$followers$followers[n] -
                  historical$followers$followers[n-30]) /
                 historical$followers$followers[n-30] * 100
    cat("30-day follower growth:", round(growth_30d, 1), "%\n")
  }
}

## End(Not run)
```

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

Description

Retrieve concurrent player count history for a specific 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:

steamAppId Integer. The Steam App ID

playerHistory Data frame with columns:

- date: Date of the data point
- avg: Average concurrent players
- median: Median concurrent players
- max: Maximum concurrent players

Examples

```
## Not run:
# Get CCU data for Counter-Strike 2
ccu_data <- vgi_insights_ccu(steam_app_id = 730)

# Plot max concurrent players over time
plot(ccu_data$playerHistory$date, ccu_data$playerHistory$max,
     type = "l",
     main = "Peak Concurrent Players",
     xlab = "Date",
     ylab = "Players")
```

```
# Calculate average peak CCU
avg_peak <- mean(ccu_data$playerHistory$max, na.rm = TRUE)
print(paste("Average peak CCU:", round(avg_peak)))

## End(Not run)
```

`vgi_insights_dau_mau` *Get Daily and Monthly Active Users Data*

Description

Retrieve daily active users (DAU) and monthly active users (MAU) data for a specific 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

You can calculate the DAU/MAU ratio from the returned data: `dau_mau_ratio = dau / mau`

The DAU/MAU ratio is a key metric for measuring player engagement:

- A ratio of 1.0 means every monthly user plays daily (perfect retention)
- A ratio of 0.5 means the average player plays 15 days per month
- A ratio of 0.1 means the average player plays 3 days per month

Industry benchmarks:

- Casual games: 0.1-0.2
- Mid-core games: 0.2-0.4
- Hardcore games: 0.4-0.6

Value

A list containing:

steamAppId Integer. The Steam App ID

playerHistory Data frame with columns:

- date: Date of the data point
- dau: Daily active users count
- mau: Monthly active users count

Examples

```
## Not run:
# Get DAU/MAU data for a game
active_players <- vgi_insights_dau_mau(steam_app_id = 730)

# Calculate DAU/MAU ratios
active_players$playerHistory$dau_mau_ratio <-
  active_players$playerHistory$dau / active_players$playerHistory$mau

# Calculate average DAU/MAU ratio
avg_ratio <- mean(active_players$playerHistory$dau_mau_ratio, na.rm = TRUE)
print(paste("Average DAU/MAU ratio:", round(avg_ratio, 3)))

# Plot DAU and MAU over time
par(mfrow = c(2, 1))
plot(active_players$playerHistory$date, active_players$playerHistory$dau,
     type = "l", col = "blue",
     main = "Daily Active Users",
     xlab = "Date", ylab = "DAU")
plot(active_players$playerHistory$date, active_players$playerHistory$mau,
     type = "l", col = "red",
     main = "Monthly Active Users",
     xlab = "Date", ylab = "MAU")

# Analyze retention trends
plot(active_players$playerHistory$date,
     active_players$playerHistory$dau_mau_ratio,
     type = "l", ylim = c(0, 1),
     main = "Player Retention (DAU/MAU Ratio)",
     xlab = "Date", ylab = "DAU/MAU Ratio")
abline(h = 0.3, col = "gray", lty = 2)  # Industry average

## End(Not run)
```

---

vgi_insights_followers *Get Follower Data for a Game*

Description

Retrieve historical follower data for a specific game on Steam, showing how many users follow the game over time.

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.

Details

Follower data indicates community engagement and interest:

- Followers receive updates about the game in their Steam activity feed
- High follower counts suggest strong community interest
- Follower growth often correlates with marketing effectiveness
- Pre-launch follower counts can predict initial sales

Value

A list containing:

steamAppId Integer. The Steam App ID

followersChange Data frame with columns:

- date: Date of the data point
- followersTotal: Total number of followers on that date
- followersChange: Change in followers from previous period

Examples

```
## Not run:
# Get follower data for a game
followers <- vgi_insights_followers(steam_app_id = 892970)

# Display current followers
current_followers <- tail(followers$followersChange, 1)$followersTotal
print(paste("Current followers:", format(current_followers, big.mark = ",")))

# Calculate growth rate
if (nrow(followers$followersChange) >= 7) {
  week_ago <- followers$followersChange[nrow(followers$followersChange) - 6, ]
  weekly_growth <- current_followers - week_ago$followersTotal
  print(paste("Weekly growth:", format(weekly_growth, big.mark = ",")))
}

# Plot follower growth
plot(followers$followersChange$date, followers$followersChange$followersTotal,
     type = "l", col = "darkgreen", lwd = 2,
     main = "Follower Growth Over Time",
     xlab = "Date", ylab = "Total Followers")

# Add daily changes as bars
par(new = TRUE)
barplot(followers$followersChange$followersChange,
        col = ifelse(followers$followersChange$followersChange > 0,
                     "lightgreen", "lightcoral"),
        border = NA, axes = FALSE, xlab = "", ylab = "")
axis(4)
mtext("Daily Change", side = 4, line = 3)

## End(Not run)
```

vgi_insights_player_regions
Get Player Geographic Distribution Data

Description

Retrieve the geographic distribution of players for a specific game by region.

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.

Details

Understanding player geographic distribution is crucial for:

- Localization decisions - Which languages to prioritize
- Server placement - Where to host game servers for optimal latency
- Marketing focus - Which regions to target with advertising
- Content scheduling - When to release updates based on player timezones
- Regional pricing - Understanding purchasing power by region

Value

A list containing:

steamAppId Integer. The Steam App ID

regions Data frame with columns:

- regionName: Geographic region (e.g., "Europe", "North America", "Asia")
- rank: Rank of the region by player percentage (1 = highest)
- percentage: Percentage of total player base from this region

Examples

```
## Not run:
# Get player regions for a game
regions <- vgi_insights_player_regions(steam_app_id = 730)

# Display regions sorted by rank
print(regions$regions)

# Find the top region
top_region <- regions$regions[regions$regions$rank == 1, ]
print(paste("Top region:", top_region$regionName,
            "with", round(top_region$percentage, 1), "% of players"))

# Create a horizontal bar chart of regions
par(mar = c(5, 8, 4, 2))  # Increase left margin for region names
barplot(regions$regions$percentage,
        names.arg = regions$regions$regionName,
        horiz = TRUE,
        las = 1,
        main = "Player Distribution by Region",
        xlab = "Percentage of Players",
        col = rainbow(nrow(regions$regions), alpha = 0.8))

# Check geographic diversity
top_3_percentage <- sum(head(regions$regions$percentage, 3))
print(paste("Top 3 regions account for",
            round(top_3_percentage, 1), "% of players"))

# Identify potential server locations
major_regions <- regions$regions[regions$regions$percentage > 10, ]
print(paste("Regions with >10% of players:",
            paste(major_regions$regionName, collapse = ", ")))

## End(Not run)
```

vgi_insights_playtime *Get Playtime Statistics for a Game*

Description

Retrieve detailed playtime statistics for a specific game, including average and median playtime, ranking information, and distribution across playtime ranges.

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.

Details

The playtime data is returned in minutes. To convert to hours, divide by 60.

Playtime ranges help understand player engagement patterns:

- "0": Players who own but never played
- "<2": Players who tried briefly (under 2 hours)
- "2-5": Short engagement
- "5-10": Moderate engagement
- "10-20": Good engagement
- "20-50": Strong engagement
- "50-100": Very engaged players
- "100-200": Highly engaged players
- "200-500": Dedicated players
- "500+": Extremely dedicated players

Value

A list containing:

steamAppId Integer. The Steam App ID

avgPlaytime Integer. Average lifetime playtime across all users (in minutes)

medianPlaytime Integer. Median lifetime playtime across all users (in minutes)

avgPlaytimeRank Integer. Rank of the game by average playtime

avgPlaytimePrct Numeric. Percentile of average playtime

playtimeRanges Data frame with columns:

- range: Character string describing the playtime range (e.g., "0", "<2", "2-5", "5-10", etc.)
- percentage: Percentage of players in this range

Examples

```
## Not run:
# Get playtime statistics for a game
playtime <- vgi_insights_playtime(steam_app_id = 730)

# Display overall statistics
print(paste("Average playtime:", round(playtime$avgPlaytime / 60, 1), "hours"))
print(paste("Median playtime:", round(playtime$medianPlaytime / 60, 1), "hours"))
print(paste("Playtime rank:", playtime$avgPlaytimeRank))
print(paste("Better than", round(100 - playtime$avgPlaytimePrct, 1), "% of games"))

# Visualize playtime distribution
if (nrow(playtime$playtimeRanges) > 0) {
  barplot(playtime$playtimeRanges$percentage,
          names.arg = playtime$playtimeRanges$range,
          main = "Player Playtime Distribution",
          xlab = "Hours Played",
          ylab = "Percentage of Players",
          col = "steelblue",
          las = 2)
}

# Calculate engaged players (20+ hours)
engaged_ranges <- c("20-50", "50-100", "100-200", "200-500", "500+")
engaged_players <- sum(playtime$playtimeRanges$percentage[
  playtime$playtimeRanges$range %in% engaged_ranges
], na.rm = TRUE)
print(paste("Engaged players (20+ hours):", round(engaged_players, 1), "%"))

## End(Not run)
```

vgi_insights_price_history

Get Price History Data for a Game

Description

Retrieve historical pricing data for a specific game across different currencies.

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. Currency code (e.g., "USD", "EUR", "GBP"). If not specified, returns price history for all currencies.
`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

If currency is specified, returns a list containing:

steamAppId Integer. The Steam App ID

currency Character. The currency code

priceChanges Data frame with columns:

- priceInitial: Full price without discount
- priceFinal: Price that the game is sold at
- firstDate: First date when this price was recorded
- lastDate: Last date when this price was active (NULL if current)

If currency is not specified, returns a list containing:

steamAppId Integer. The Steam App ID

price List of price histories for each currency

Examples

```
## Not run:
# Get price history for a game in USD
usd_history <- vgi_insights_price_history(
  steam_app_id = 730,
  currency = "USD"
)

# Calculate discount percentage for each price period
if (nrow(usd_history$priceChanges) > 0) {
  usd_history$priceChanges$discount_pct <-
    round((1 - usd_history$priceChanges$priceFinal /
           usd_history$priceChanges$priceInitial) * 100, 1)
}

# Get price history for all currencies
all_prices <- vgi_insights_price_history(steam_app_id = 730)

# Find all currencies where the game is available
currencies <- sapply(all_prices$price, function(x) x$currency)
print(paste("Available in", length(currencies), "currencies"))

# Identify sales periods (where priceFinal < priceInitial)
sales <- usd_history$priceChanges[
  usd_history$priceChanges$priceFinal < usd_history$priceChanges$priceInitial,
]
print(paste("Number of sale periods:", nrow(sales)))

## End(Not run)
```

vgi_insights_revenue *Get Revenue Insights from Video Game Insights*

Description

Retrieve revenue history data for a specific game.

Usage

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

Arguments

`steam_app_id`	Integer or character. 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

The new API provides revenue history as changes rather than absolute values. Each entry shows the revenue change from the previous period.

Value

A tibble containing revenue history with columns:

steamAppId Integer. The Steam App ID

date Date. The date of the revenue data

revenueChange Numeric. Revenue change amount

revenueChangePercent Numeric. Revenue change percentage

Examples

```
## Not run:
# Get revenue history for a game
revenue_data <- vgi_insights_revenue(steam_app_id = 892970)

# Plot revenue changes over time
plot(revenue_data$date, revenue_data$revenueChange,
     type = "l",
     main = "Revenue Changes Over Time",
     xlab = "Date",
     ylab = "Revenue Change")

## End(Not run)
```

`vgi_insights_reviews` *Get Review Analytics from Video Game Insights*

Description

Retrieve review history data for a specific game, including positive and negative review counts.

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.

Details

The new API provides both incremental changes and cumulative totals for reviews. You can calculate the overall rating percentage from the totals: `rating = positiveReviewsTotal / (positiveReviewsTotal + negativeReviewsTotal) * 100`

Value

A data frame containing review history with columns:

steamAppId Integer. The Steam App ID

date Date. The date of the review data

positiveReviewsChange Integer. New positive reviews since last period

positiveReviewsTotal Integer. Total cumulative positive reviews

negativeReviewsChange Integer. New negative reviews since last period

negativeReviewsTotal Integer. Total cumulative negative reviews

Examples

```
## Not run:
# Get review history for a game
reviews <- vgi_insights_reviews(steam_app_id = 892970)

# Calculate current overall rating
latest <- tail(reviews, 1)
rating <- latest$positiveReviewsTotal /
          (latest$positiveReviewsTotal + latest$negativeReviewsTotal) * 100
print(paste("Overall rating:", round(rating, 1), "%"))

# Plot review trends
plot(reviews$date, reviews$positiveReviewsChange,
     type = "l", col = "green",
     main = "Daily Review Trends",
     xlab = "Date", ylab = "New Reviews")
lines(reviews$date, reviews$negativeReviewsChange, col = "red")
legend("topright", c("Positive", "Negative"),
       col = c("green", "red"), lty = 1)

# Find review bombs (days with unusual negative reviews)
avg_negative <- mean(reviews$negativeReviewsChange, na.rm = TRUE)
sd_negative <- sd(reviews$negativeReviewsChange, na.rm = TRUE)
review_bombs <- reviews[reviews$negativeReviewsChange > avg_negative + 2*sd_negative, ]

## End(Not run)
```

vgi_insights_units *Get Units Sold Data for a Game*

Description

Retrieve historical units sold data for a specific game.

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.

Details

The new API provides both incremental changes and cumulative totals for units sold. This makes it easy to track both growth rates and absolute numbers.

Value

A data frame containing units sold history with columns:

steamAppId Integer. The Steam App ID

date Date. The date of the data point

unitsSoldChange Integer. Units sold change from previous period

unitsSoldTotal Integer. Total cumulative units sold

Examples

```
## Not run:
# Get units sold history for a game
units_data <- vgi_insights_units(steam_app_id = 730)

# Plot cumulative units sold over time
plot(units_data$date, units_data$unitsSoldTotal,
     type = "l", main = "Total Units Sold Over Time",
     xlab = "Date", ylab = "Total Units")

# Calculate daily sales for recent period
recent_data <- tail(units_data, 30)
daily_sales <- mean(recent_data$unitsSoldChange, na.rm = TRUE)
print(paste("Average daily sales (last 30 days):", round(daily_sales)))
```

```
# Find peak sales day
peak_day <- units_data[which.max(units_data$unitsSoldChange), ]
print(paste("Peak sales:", peak_day$unitsSoldChange, "on", peak_day$date))

## End(Not run)
```

vgi_insights_wishlists *Get Wishlist Data for a Game*

Description

Retrieve historical wishlist data for a specific game, showing how many users have the game on their wishlist over time.

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.

Details

Wishlist data is a key indicator of interest and potential future sales:

- High wishlist numbers indicate strong market interest
- Wishlist spikes often follow marketing campaigns or announcements
- Conversion rate from wishlist to purchase typically ranges from 5-20%
- Wishlist trends can predict launch day sales

Value

A list containing:

steamAppId Integer. The Steam App ID

wishlistChanges Data frame with columns:

- date: Date of the data point
- wishlistsTotal: Total number of wishlists on that date
- wishlistsChange: Change in wishlists from previous period

Examples

```
## Not run:
# Get wishlist data for a game
wishlists <- vgi_insights_wishlists(steam_app_id = 892970)

# Calculate total wishlists and recent trend
current_wishlists <- tail(wishlists$wishlistChanges, 1)$wishlistsTotal
print(paste("Current wishlists:", format(current_wishlists, big.mark = ",")))

# Calculate 30-day growth
if (nrow(wishlists$wishlistChanges) >= 30) {
  thirty_days_ago <- wishlists$wishlistChanges[nrow(wishlists$wishlistChanges) - 29, ]
  growth <- current_wishlists - thirty_days_ago$wishlistsTotal
  growth_pct <- (growth / thirty_days_ago$wishlistsTotal) * 100
  print(paste("30-day growth:", format(growth, big.mark = ","),
              sprintf("(%.1f%%)", growth_pct)))
}

# Plot wishlist trend
plot(wishlists$wishlistChanges$date, wishlists$wishlistChanges$wishlistsTotal,
     type = "l", col = "blue", lwd = 2,
     main = "Wishlist Trend Over Time",
     xlab = "Date", ylab = "Total Wishlists")

# Identify major wishlist spikes
avg_change <- mean(abs(wishlists$wishlistChanges$wishlistsChange), na.rm = TRUE)
spikes <- wishlists$wishlistChanges[
  wishlists$wishlistChanges$wishlistsChange > avg_change * 3,
]
if (nrow(spikes) > 0) {
  print("Major wishlist spikes detected on:")
  print(spikes[, c("date", "wishlistsChange")])
}

## End(Not run)
```

vgi_player_overlap *Get Player Overlap Data*

Description

Retrieve player overlap data showing which other games are played by players of a specific game. This helps identify similar games and player preferences.

Usage

```
vgi_player_overlap(
  steam_app_id,
  limit = 10,
  offset = 0,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

`steam_app_id` Integer. The Steam App ID of the main game.

`limit` Integer. Maximum number of overlapping games to return (default 10).

`offset` Integer. Number of records to skip for pagination (default 0).

`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

Player overlap data is valuable for:

- Competitive analysis - Identify direct competitors
- Marketing - Find games with similar audiences for cross-promotion
- Game design - Understand what other games your players enjoy
- Platform strategy - Identify bundle opportunities

The overlap index is particularly useful:

- Index > 2.0: Strong overlap, very similar audience
- Index 1.5-2.0: Moderate overlap, some audience similarity
- Index 1.0-1.5: Slight overlap, minimal similarity
- Index < 1.0: Below average overlap

Value

A list containing:

steamAppId Integer. The Steam App ID of the main game

playerOverlaps Data frame with columns for each overlapping game:

- steamAppId: ID of the overlapping game
- medianPlaytime: Median hours played of main game by overlap players
- unitsSoldOverlap: Number of players who own both games
- unitsSoldOverlapPercentage: Percent of main game owners who own this game
- unitsSoldOverlapIndex: How much more likely to own vs average Steam user
- mauOverlap: Monthly active users who play both games
- mauOverlapPercentage: Percent of main game MAU who play this game
- mauOverlapIndex: How much more likely to play vs average Steam user
- wishlistOverlap: Number who wishlist both games
- wishlistOverlapPercentage: Percent of main game wishlisters who wishlist this
- wishlistOverlapIndex: How much more likely to wishlist vs average

Examples

```
## Not run:
# Get player overlap for a game
overlap <- vgi_player_overlap(steam_app_id = 892970, limit = 20)

# Find games with strongest overlap
strong_overlap <- overlap$playerOverlaps[
  overlap$playerOverlaps$unitsSoldOverlapIndex > 2.0,
]
print(paste("Games with strong overlap:", nrow(strong_overlap)))

# Analyze competitor landscape
competitors <- head(overlap$playerOverlaps[
  order(-overlap$playerOverlaps$unitsSoldOverlapPercentage),
], 5)
print("Top 5 competitors by player overlap:")
print(competitors[, c("steamAppId", "unitsSoldOverlapPercentage",
                      "unitsSoldOverlapIndex")])

# Find games where overlap players are highly engaged
engaged_overlap <- overlap$playerOverlaps[
  overlap$playerOverlaps$medianPlaytime > 50,
]
print(paste("Games where overlap players spend 50+ hours:",
            nrow(engaged_overlap)))

# Calculate total addressable market from overlap
total_overlap_players <- sum(overlap$playerOverlaps$unitsSoldOverlap)
print(paste("Total unique players across all overlaps:",
            format(total_overlap_players, big.mark = ",")))

## End(Not run)
```

vgi_publisher_games *Get Game IDs by Publisher*

Description

Retrieve a list of Steam App IDs for all games by a specific publisher.

Usage

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

Arguments

`company_id`	Integer. The publisher's company ID.
`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 now returns only game IDs instead of full game metadata. To get detailed information about each game, use `vgi_game_metadata()` with the returned IDs.

This approach is more efficient when you only need to:

- Count the number of games by a publisher
- Check if a publisher released a specific game
- Get a subset of games for detailed analysis

Value

A numeric vector of Steam App IDs for games published by this company.

Examples

```
## Not run:
# Get all game IDs for a publisher
game_ids <- vgi_publisher_games(company_id = 13190)

# Count games by this publisher
cat("Total games published:", length(game_ids), "\n")

# Get detailed info for the first 5 games
if (length(game_ids) > 0) {
  first_five <- head(game_ids, 5)
  game_details <- lapply(first_five, vgi_game_metadata)

  # Display game names
  for (game in game_details) {
    cat(game$name, "(", game$steamAppId, ")\n")
  }
}

# Analyze publisher's portfolio
if (length(game_ids) > 0) {
  # Get metadata for all games
  all_games <- vgi_game_metadata_batch(game_ids)

  # Group by genre
  genre_counts <- table(unlist(lapply(all_games$genres, function(x) x)))
  print("Publisher's genre distribution:")
  print(sort(genre_counts, decreasing = TRUE))
}

# Find publisher's most successful games
if (length(game_ids) > 10) {
  # Get revenue data for top games
  revenues <- lapply(head(game_ids, 10), function(id) {
    tryCatch({
      rev_data <- vgi_insights_revenue(id)
      list(id = id, revenue = rev_data$revenueTotal)
    }, error = function(e) NULL)
  })

  # Filter out NULLs and sort by revenue
  revenues <- revenues[!sapply(revenues, is.null)]
  revenues <- revenues[order(sapply(revenues, function(x) x$revenue),
                             decreasing = TRUE)]
}

## End(Not run)
```

vgi_publisher_info *Get Publisher Information*

Description

Retrieve detailed information about a specific video game publisher.

Usage

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

Arguments

company_id	Integer. The publisher's company ID.
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

Publisher information can be used to:

- Research company backgrounds and portfolios
- Analyze publisher market share
- Identify publishers specializing in certain genres
- Track publisher growth and success metrics

Value

A list containing publisher information with fields:

companyId Integer. The company ID

name Character. Publisher name

foundedDate Character. Date the company was founded

headquarters Character. Company headquarters location

employeeCount Integer. Number of employees

website Character. Company website URL

description Character. Company description

totalGames Integer. Total number of games published

activeGames Integer. Number of currently active games

Examples

```
## Not run:
# Get information about a publisher
pub_info <- vgi_publisher_info(company_id = 13190)

# Display publisher details
cat("Publisher:", pub_info$name, "\n")
cat("Founded:", pub_info$foundedDate, "\n")
cat("Total Games Published:", pub_info$totalGames, "\n")
cat("Currently Active Games:", pub_info$activeGames, "\n")

# Analyze publisher size
if (!is.null(pub_info$employeeCount)) {
  if (pub_info$employeeCount > 1000) {
    cat("This is a major publisher\n")
  } else if (pub_info$employeeCount > 100) {
    cat("This is a mid-sized publisher\n")
  } else {
    cat("This is a boutique publisher\n")
  }
}

## End(Not run)
```

vgi_publisher_list *Get Complete Publisher List*

Description

Retrieve a list of all game publishers in the Video Game Insights database.

Usage

```
vgi_publisher_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.

Details

This endpoint is useful for:

- Discovering all tracked publishers
- Building publisher selection interfaces
- Finding publisher IDs for further queries
- Analyzing the publishing landscape

Value

A data frame with columns:

id Integer. The company ID

name Character. The publisher name

Examples

```
## Not run:
# Get all publishers
all_pubs <- vgi_publisher_list()
cat("Total publishers:", nrow(all_pubs), "\n")

# Search for major publishers
ea_pubs <- all_pubs[grep("Electronic Arts", all_pubs$name, ignore.case = TRUE), ]
print(ea_pubs)

# Find publisher ID by exact name
pub_id <- all_pubs$id[all_pubs$name == "Ubisoft"]
if (length(pub_id) > 0) {
  cat("Ubisoft ID:", pub_id, "\n")

  # Get more info about this publisher
  ubi_info <- vgi_publisher_info(pub_id)
  print(ubi_info)
}

# Analyze publisher ecosystem
# Find self-published developers
all_devs <- vgi_developer_list()
self_published <- intersect(all_pubs$name, all_devs$name)
cat("Companies that both develop and publish:", length(self_published), "\n")

# Find publishers with "Games" in name
games_pubs <- all_pubs[grep("Games", all_pubs$name), ]
cat("Publishers with 'Games' in name:", nrow(games_pubs), "\n")

# Export for analysis
write.csv(all_pubs, "vgi_publishers.csv", row.names = FALSE)

## 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,
  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.
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,
  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.
`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. Since the new API doesn't have a dedicated search endpoint, this function fetches the game list and performs client-side filtering.

Usage

```
vgi_search_games(
  query,
  limit = 10,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

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.

Details

This function now uses the `/games/game-list` endpoint and filters results locally. For better performance with large datasets, consider caching the game list.

Value

A tibble containing search results with game information including Steam App ID 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_steam_market_data` *Get Steam Market Data Analytics*

Description

Retrieve Steam market analytics including global statistics and trends.

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.

Details

This endpoint provides high-level market analytics for the entire Steam platform. Use this data to:

- Understand overall market size and trends
- Analyze price distributions for competitive positioning
- Track new release velocity
- Identify popular genres and tags
- Benchmark your game against market averages

Value

A list containing Steam market analytics data with components:

totalGames Integer. Total number of games on Steam

totalRevenue Numeric. Total revenue across all games

totalUnitsSold Numeric. Total units sold across all games

averagePrice Numeric. Average game price

medianPrice Numeric. Median game price

totalDevelopers Integer. Total number of developers

totalPublishers Integer. Total number of publishers

gamesReleasedLast30Days Integer. Number of games released in last 30 days

gamesReleasedLast365Days Integer. Number of games released in last year

topGenres Data frame with genre statistics

topTags Data frame with tag statistics

priceDistribution Data frame with price range distributions

Examples

```
## Not run:
# Get Steam market analytics
market_data <- vgi_steam_market_data()

# Display key market metrics
cat("Total Steam Games:", format(market_data$totalGames, big.mark = ","), "\n")
cat("Total Revenue: $", format(market_data$totalRevenue, big.mark = ","), "\n")
cat("Average Game Price: $", round(market_data$averagePrice, 2), "\n")
cat("Games Released Last 30 Days:", market_data$gamesReleasedLast30Days, "\n")

# Analyze top genres
if (!is.null(market_data$topGenres)) {
  print("Top 5 Genres by Game Count:")
  print(head(market_data$topGenres, 5))
}

# Examine price distribution
if (!is.null(market_data$priceDistribution)) {
  barplot(market_data$priceDistribution$count,
          names.arg = market_data$priceDistribution$priceRange,
          main = "Steam Games Price Distribution",
          xlab = "Price Range",
          ylab = "Number of Games",
          col = "steelblue")
}

# Calculate market concentration
top_10_percent_games <- market_data$totalGames * 0.1
cat("The top 10% of games (", round(top_10_percent_games),
    " games) likely generate 80%+ of revenue\n", sep = "")

## End(Not run)
```

vgi_top_countries *Get Top Countries by Player Count*

Description

Retrieve the top countries by player count for a specific game, showing where the game's player base is concentrated geographically.

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.

Details

This endpoint provides insights into:

- Geographic distribution of your player base
- Key markets for localization efforts
- Regional marketing opportunities
- Server location planning

Countries are ranked by total player count, typically showing the top 20-50 countries depending on the game's distribution.

Value

A data frame with columns:

country Character. Two-letter country code (ISO 3166-1 alpha-2)

countryName Character. Full country name

playerCount Integer. Number of players from this country

percentage Numeric. Percentage of total player base

rank Integer. Country rank by player count

Examples

```
## Not run:
# Get top countries for a game
top_countries <- vgi_top_countries(steam_app_id = 730)

# Display top 10 countries
head(top_countries, 10)

# Calculate cumulative percentage
top_countries$cumulative_pct <- cumsum(top_countries$percentage)

# Find how many countries make up 80% of players
countries_80pct <- which(top_countries$cumulative_pct >= 80)[1]
cat("80% of players come from top", countries_80pct, "countries\n")

# Create a bar chart of top 10 countries
top10 <- head(top_countries, 10)
barplot(top10$percentage,
        names.arg = top10$countryName,
        las = 2,
        main = "Top 10 Countries by Player %",
        ylab = "Percentage of Players",
        col = "steelblue")

# Check for specific regions
eu_countries <- c("DE", "FR", "GB", "IT", "ES", "PL", "NL", "SE", "BE", "AT")
eu_players <- sum(top_countries$percentage[top_countries$country %in% eu_countries])
cat("EU player percentage:", round(eu_players, 1), "%\n")

# Identify emerging markets
emerging <- top_countries[top_countries$rank > 10 & top_countries$percentage > 1, ]
cat("Emerging markets (rank >10 but >1%):", nrow(emerging), "\n")
print(emerging)

## End(Not run)
```

`vgi_top_games` *Get Top Games from Video Game Insights*

Description

Retrieve top games ranked by various metrics including revenue, units sold, concurrent users (CCU), daily active users (DAU), or followers.

Usage

```
vgi_top_games(
  metric,
  platform = "all",
  start_date = NULL,
  end_date = NULL,
  limit = 100,
  auth_token = Sys.getenv("VGI_AUTH_TOKEN"),
  headers = list()
)
```

Arguments

`metric`	Character string. The metric to rank games by. Must be one of: "revenue", "units", "ccu", "dau", or "followers".
`platform`	Character string. Platform to filter by. Options are: "steam", "playstation", "xbox", "nintendo", or "all". Defaults to "all".
`start_date`	Date or character string. Start date for the ranking period in YYYY-MM-DD format. Optional.
`end_date`	Date or character string. End date for the ranking period in YYYY-MM-DD format. Optional.
`limit`	Integer. Maximum number of results to return. Defaults to 100.
`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 containing top games with columns:

- steam_app_id: The Steam App ID
- name: Game name (when available)
- rank: Rank for the specified metric (1 = best)
- percentile: Percentile ranking (0-100)
- value: Same as percentile (for backwards compatibility)

Examples

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

# Get top 10 games by revenue
top_revenue <- vgi_top_games("revenue", limit = 10)
print(top_revenue)

# Get top Steam games by CCU for a specific date range
top_ccu_steam <- vgi_top_games(
  metric = "ccu",
  platform = "steam",
  start_date = "2024-01-01",
  end_date = "2024-01-31",
  limit = 50
)

## End(Not run)
```

`vgi_top_wishlist_countries`

Get Top Countries by Wishlist Count

Description

Retrieve the top countries by wishlist count for a specific game, showing where potential future players are concentrated.

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.

Details

Wishlist geographic data is valuable for:

- Pre-launch marketing focus
- Identifying high-interest regions
- Planning regional promotions
- Localization priorities for upcoming content
- Predicting launch day geographic distribution

Compare wishlist distribution with actual player distribution to identify untapped markets or conversion opportunities.

Value

A data frame with columns:

country Character. Two-letter country code (ISO 3166-1 alpha-2)

countryName Character. Full country name

wishlistCount Integer. Number of wishlists from this country

percentage Numeric. Percentage of total wishlists

rank Integer. Country rank by wishlist count

Examples

```
## Not run:
# Get top wishlist countries for a game
wishlist_countries <- vgi_top_wishlist_countries(steam_app_id = 892970)

# Display top 10 countries
head(wishlist_countries, 10)

# Compare with actual player distribution
player_countries <- vgi_top_countries(steam_app_id = 892970)

# Merge to compare wishlist vs player percentages
comparison <- merge(wishlist_countries, player_countries,
                   by = "country", suffixes = c("_wishlist", "_player"))

# Calculate conversion potential
comparison$conversion_rate <- comparison$percentage_player / comparison$percentage_wishlist
comparison <- comparison[order(comparison$conversion_rate), ]

# Find underperforming countries (high wishlist, low players)
underperforming <- comparison[comparison$conversion_rate < 0.5, ]
cat("Countries with low wishlist conversion:\n")
print(underperforming[, c("countryName_wishlist", "percentage_wishlist",
                         "percentage_player", "conversion_rate")])

# Calculate regional interest
asia_countries <- c("CN", "JP", "KR", "TW", "HK", "SG", "TH", "ID", "MY", "PH")
asia_wishlist_pct <- sum(wishlist_countries$percentage[
  wishlist_countries$country %in% asia_countries])
cat("Asia wishlist percentage:", round(asia_wishlist_pct, 1), "%\n")

# Visualize top 10 wishlist countries
top10 <- head(wishlist_countries, 10)
barplot(top10$percentage,
        names.arg = top10$countryName,
        las = 2,
        main = "Top 10 Countries by Wishlist %",
        ylab = "Percentage of Wishlists",
        col = "darkgreen")

## 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,
  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.
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,
  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.
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