Starcraft2Metrics Documentation

Release 0.1.0

Matthew Johnson

Dec 18, 2019

Contents

Starcraft2Metrics offers metrics calculated from replay files using the *SC2Reader* parser.

The end goal of Starcraft2Metrics is to offer the user a visualization of the trend of each metric that is calculated in order for the user to see if there has been improvement over a period of time. This information can be used to locate weak points in a player's skill and improve that area.

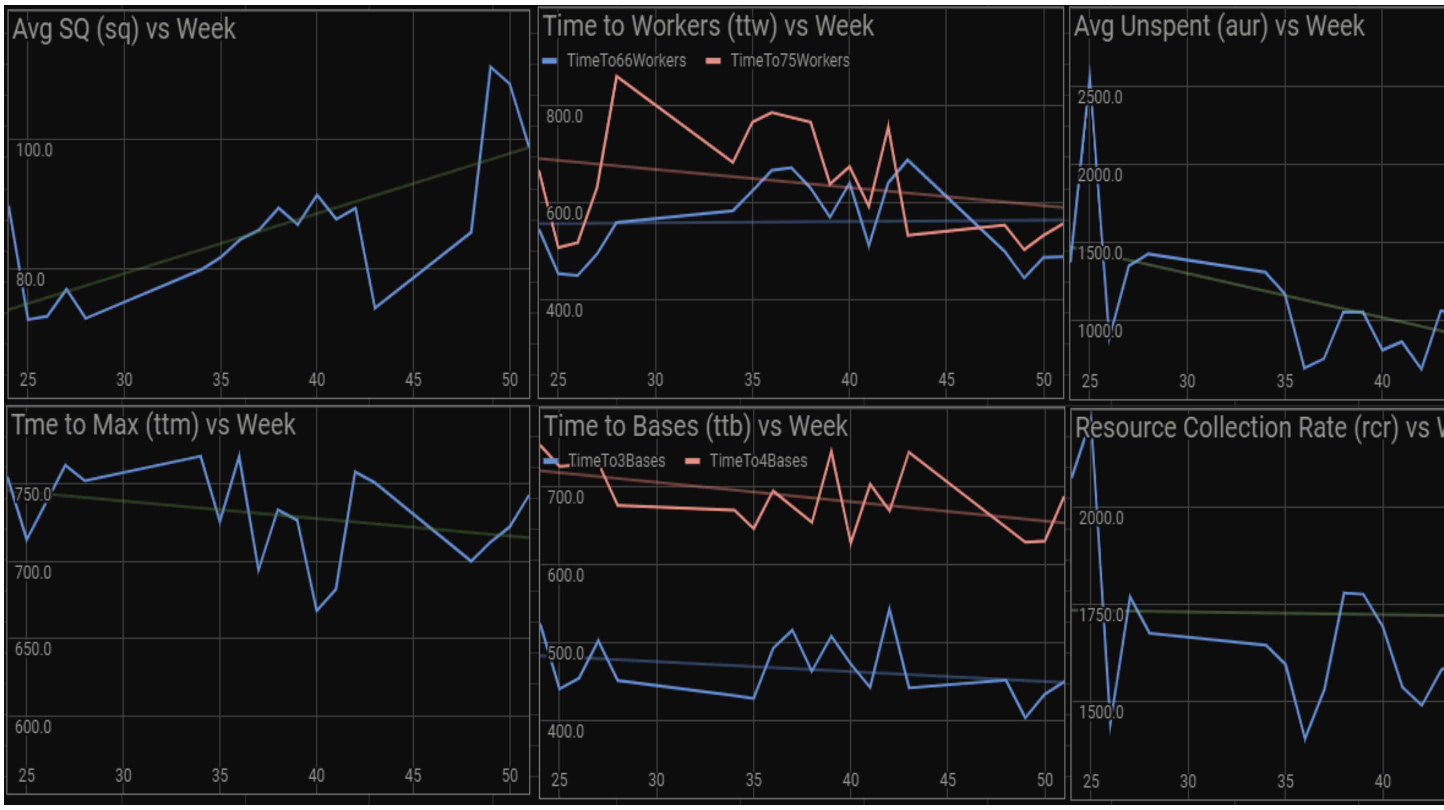

Fig. 1: Example Trends derived from Metric data.

CHAPTER 1

SC2ReplayParser

The SC2ReplayParser is a command line utility that will gather metric data from a directory of replays. The parser will output a csv file that contains a number of useful metrics along with metadata about each replay. The metric data can then be used to derive trends.

Replay metadata:

- Replay Name
- Date
- Map
- Race Matchup
- Game Length
- Game Type
- Is Ladder?

Metrics offered:

- Time to Max (ttm)
- Time to 3 bases (ttb3)
- Time to 4 bases (ttb4)
- Time to 66 workers (ttw66)
- Time to 75 workers (ttw75)
- Average APM (apm)
- Average Spending Quotient (sq)
- Average Spending Quotient Pre-Max Supply (sq-max)
- Average Unspent Resources (aur)
- Average Unspent Resources Pre-Max Supply (aur-max)

- Average Resource Collection Rate (rcr)
- Average Resource Collection Rate Pre-Max Supply (rcr-max)
- Time spent Supply Capped (sc)

CHAPTER 2

SC2Reader

SC2Reader is a Starcraft II Replay parser that is hosted publicly on Github [source]. This module is relied upon to parse the necessary data out of Starcraft II replay files.

CHAPTER 3

Reference Pages

3.1 `metrics` — sc2metric

Metrics gathered from Starcraft 2 replay data.

class `metrics.sc2metric.`**`Sc2MetricAnalyzer`**

A class that derives useful metrics from replay files

This class is an extension to the `sc2reader` player class. It is meant to be used to offer useful metric data to each player in a Starcraft II replay. The class contains raw data gathered from each replay to derive its metrics. The raw data is filled from the `plugins` associated with this package.

`army_created = None`

A list of *`SupplyCount`* for each army supply created.

`workers_created = None`

A list of *`SupplyCount`* for each worker created.

`supply_created = None`

A list of *`SupplyCount`* for each unit created.

`bases_created = None`

A list of *`BaseCount`* for each base created.

`supply = None`

A list of *`FoodCount`* for each unit and supply provider created.

`resources = None`

A list of *`ResourceCount`* for resources collected and resources unspent every 10 seconds.

`avg_apm = None`

The player's average apm

`avg_spm = None`

The player's average spm

`metrics``()`

A dictionairy of basic metrics.

`first_max`()

Returns the time at which the player first reached max supply.

`avg_sq`()

Average Spending Quotient

Calculates the average Spending Quotient (SQ).

Returns The average Spending Quotient (SQ).

Return type int

`avg_sq_at_time`(*time_s*)

Average Spending Quotient up to a specified time

Calculates the average Spending Quotient (SQ) up until the specified time.

Parameters **`time_s`** (*`int`*) – The time in the replay to stop calculating spending quotient.

Returns The average Spending Quotient (SQ) up until the specified time.

Return type int

`avg_sq_pre_max`()

Average Spending Quotient before max supply

Calculates the average Spending Quotient (SQ) up until the player first reaches max supply.

Returns

The average Spending Quotient (SQ) up until the player first maxes.

Return type int

`aur`()

Average Unspent Resources

Calculates the Average Unspent Resources (AUR) during the game.

Returns The Average Unspent Resources (AUR)

Return type int

`aur_at_time`(*time_s*)

Averague Unspent Resources up to a specified time

Calculates the Average Unspent Resources (AUR) up until the specified time.

Parameters **`time_s`** (*`int`*) – The time in the replay to stop calculating average unspent resources.

Returns

The Average Unspent Resources (AUR) up until the specified time.

Return type int

`aur_pre_max`()

Average Unspent Resources before max supply

Calculates the Average Unspent Resources (AUR) up until the player first reaches max supply.

Returns

The Average Unspent Resources (AUR) up until the player first maxes.

Return type int

`avg_rcr`()

Average Resource Collection Rate

Calculates the average Resource Collection Rate (RCR) during the game.

Returns The average Resource Collection Rate (RCR)

Return type int

`avg_rcr_at_time`(*time_s*)

Average Resource Collection Rate up to a specified time

Calculates the average Resource Collection Rate (RCR) up until the specified time.

Parameters **`time_s`** (*`int`*) – The time in the replay to stop calculating average resource collection rate.

Returns

The average Resource Collection Rate (RCR) up until the specified time.

Return type int

`avg_rcr_pre_max`()

Average Resource Collection Rate before max supply

Calculates the average Resource Collection Rate (RCR) up until the player first reaches max supply.

Returns

The average Resource Collection Rate (RCR) up until the player first maxes.

Return type int

`supply_capped`()

Time spent supply capped

Determines the amount of time the player was supply capped this game. Supply capped is acknowledged when supply of units = supply created when supply created < 200.

Returns The amount of time spent supply capped.

Return type int

`supply_at_time`(*real_time_s*)

The current supply used at the specified time.

Calculates the current supply used at the time.

Parameters **`real_time_s`** (*`int`*) – The time (in seconds) of the replay to count supply.

Returns The current amount of supply used in the given time.

Return type int

`first_time_to_supply`(*supply*)

The first time that the specified supply was reached.

Finds the time when the specified supply was reached.

Parameters **`supply`** (*`int`*) – The supply desired.

Returns The time when the supply was first reached.

Return type int

`workers_created_at_time`(*time_s*)

Number of workers created up to the specified time

Determines the total number of workers created at the specified time. This function is accumulative and does not handle workers lost.

Parameters **`time_s`** (*`int`*) – The time (in seconds) of the replay to count workers.

Returns The number of workers created.

Return type int

`army_created_at_time`(*game_time_s*)

The army supply created up to a specified time

Calculate the total army supply created at the specified time.

Parameters **`game_time_s`** (*`int`*) – The time (in seconds) of the replay to count army supply.

Returns The amount of army supply created in the given time.

Return type int

`supply_created_at_time`(*real_time_s*)

The total supply created up to a specified time

Calculate the total supply created at the specified time.

Parameters **`real_time_s`** (*`int`*) – The time (in seconds) of the replay to count supply.

Returns The amount of supply created in the given time.

Return type int

`time_to_workers_created`(*worker_count*)

The time at which the number of workers had been created

Finds the time that the specified number of workers have been created. This does not account for loss of workers.

Parameters **`worker_count`** (*`int`*) – The number of workers to find the time created.

Returns The time at which the total number of workers were created.

Return type int

`time_to_supply_created`(*supply_count*)

The time at which the specified supply had been created

Finds the time when the specified supply was created. This does not take into account supply lost.

Parameters **`supply_count`** (*`int`*) – The supply desired.

Returns The time when the supply was created in the replay.

Return type int

`time_to_supply_created_max_workers`(*supply_count*, *max_workers_counted*)

The time at which the specified supply had been created

Finds the time when the specified supply was reached, but only counts a specified number of workers towards that supply created count. This does not take into account supply lost.

Parameters

- **`supply_count`** (*`int`*) – The supply desired.

- **max_workers_counted** (*int*) – The maximum number of workers to count towards the supply created.

Returns The time when the supply was created in the replay.

Return type int

time_to_bases_created(*base_count*)
The time at which the number of bases had been created

Finds the time when the specified number of bases has been reached. This does not take into account any bases lost.

Parameters **base_count** (*int*) – The number of bases desired.

Returns The time when the number of bases had been created.

Return type int

3.1.1 Containers

class metrics.metric_containers.**FoodCount**(*second*, *supply_used*, *supply_made*)
Container for tracking supply at a given second.

Parameters

- **second** (*int*) – The second in the game.
- **supply_used** (*int*) – The amount of supply used.
- **supply_made** (*int*) – The amount of supply made from supply buildings.

class metrics.metric_containers.**BaseCount**(*second*)
Container for the second a base was created in a game.

Parameters **second** (*int*) – The second a base was created.

class metrics.metric_containers.**SupplyCount**(*second*, *total_supply*, *unit_supply*, *is_worker*)
Container for the supply of a player at a specified second in the game. Also includes the supply of the last unit created.

Parameters

- **second** (*int*) – The second in the game.
- **total_supply** (*int*) – The total supply used as this second in the game.
- **unit_supply** (*int*) – The supply of the last unit made.
- **is_worker** (*bool*) – Boolean that indicates whether the last unit made was a worker or not.

class metrics.metric_containers.**ResourceCount**(*second*, *res_col*, *res_unspent*)
Container for the resource collection rate and unspent at a specified second in the game.

Parameters

- **second** (*int*) – The second in the game.
- **res_col** (*int*) – The resource collection rate at this second in the game.
- **res_unspent** (*int*) – The unspent resources at this second in the game.

3.2 `plugins` — sc2reader Plugins

Plugins used to gather the raw data for the *`metrics.sc2metric.Sc2MetricAnalyzer`*.

class `metrics.plugins.supply.`**`SupplyTracker`**

Builds `player.metrics.supply` array made of *`FoodCount`*. The `metrics` being of the type *`Sc2MetricAnalyzer`*. The supply is tracked every time a new supply unit or supply building is made or dies/is destroyed.

class `metrics.plugins.supply_created.`**`SupplyCreatedTracker`**

Builds `player.metrics.army_created`, `player.metrics.workers_created`, and `player.metrics.supply_created` arrays made of *`SupplyCount`*. The `metrics` being of the type *`Sc2MetricAnalyzer`*. The supplies are tracked whenever a unit is created. The unit's supply and a cumulative supply count of the army, workers, and total supply are tracked for the corresponding second.

class `metrics.plugins.bases_created.`**`BasesCreatedTracker`**

Builds `player.metrics.bases_created` array made of *`BaseCount`*. The `metrics` being of the type *`Sc2MetricAnalyzer`*. The bases are tracked every time a *Nexus*, *CommandCenter*, or *Hatchery* is completed.

class `metrics.plugins.resources.`**`ResourceTracker`**

Builds `player.metrics.resources` array made of *`ResourceCount`*. The `metrics` being of the type *`Sc2MetricAnalyzer`*. The resources tracked are the unspent resources (minerals + vespene) and the resource collection rate (minerals + vespene).

class `metrics.plugins.apm.`**`APMTracker`**

Provides `player.metrics.avg_apm` which is defined as the sum of any Selection, ControlGroup, or Command event issued by the player divided by the number of seconds played by the player (not necessarily the whole game) multiplied by 60. APM is 0 for games under 1 minute in length.

*Note: APM is generally calculated starting after an initial time. For example, APM in Brood War was calculated after the initial 150 seconds of game time. Therefore, actions before 150 seconds were not counted towards actual APM.

**Note: These APM calculations include the actions taken before 150 seconds, but they do not include that 150 seconds of time, which is standard among most APM calculators.

3.3 SC2 Replay Parser

3.4 `util` — Utility Functions

Contains a number of useful functions.

`metrics.util.`**`convert_to_gametime_r`**(*replay*, *real_time_s*)

Converts the real time of a point in the replay to the internally calculated game time.

Parameters

- **`replay`** (*`sc2reader.replay`*) – The sc2reader replay object from which to convert the real time to.
- **`real_time_s`** (*`int`*) – The real time in the replay in seconds.

Returns The game time in seconds.

Return type int

`metrics.util.`**`convert_to_realtime_r`**(*replay*, *game_time_s*)

Converts the internally calculated game time of a point in the replay to the real time, as can be seen when watching a replay.

Parameters

- **`replay`** (*`sc2reader.replay`*) – The sc2reader replay object from which to convert the real time to.
- **`game_time_s`** (*`int`*) – The game time internally calculated in the replay in seconds.

Returns The real time in seconds.

Return type int

`metrics.util.`**`is_hallucinated`**(*unit*)

A special function used to bypass a bug found in `sc2reader.Data.Unit.hallucinated`, where the hallucinated property was not returning a correct boolean.

Parameters **`unit`** (*`sc2reader.Data.Unit`*) – The `Unit` object.

Returns True if the unit is hallucinated, False if it was not.

Return type bool

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index