Tarrasque Documentation

Release 0.1

Laurie Clark-Michalek

January 04, 2014

Contents

1 An Introduction to Tarrasque **3**
1.1 Tarrasque concepts for people who know what an ehandle is 4
1.2 Tarrasque concepts for people who don't know what an ehandle is 4

2 Guides **5**
2.1 Analysing game end states 5

3 API **7**
3.1 Stream Binding 7
3.2 Creep Manager 9
3.3 Dota Entity 9
3.4 Player 10
3.5 Game Info 11
3.6 Ability 13
3.7 Base NPC 14
3.8 Hero 15
3.9 Game Events 16
3.10 Combat Log 16
3.11 Item 17

4 Indices and tables **19**

Python Module Index **21**

Contents:

CHAPTER 1

An Introduction to Tarrasque

Tarrasque is a library, build around Skadi, to allow the easy and straightforward analysis of Dota 2 replays. While Skadi provides only the raw data, Tarrasque allows you to deal in objects and relationships. A comparison will show this best.

This code uses Skadi to print out the names of the players in the replay, along with the name of the hero they are playing

```
import io
from skadi.engine import world as w
from skadi.replay import demo as rd

demo = rd.construct("./demo.dem")
for tick, string_tables, world in demo.stream(tick=5000):
    ehandle, player_resource = world.find_by_dt(player_resource_dt)

    for i in range(31):
        player_name_key = ("DT_DOTA_PlayerResource", "m_iszPlayerNames.%40s" % i)
        player_name = player_resource[player_name_key]
        if not player_name:
            break
        hero_ehandle_key = ("DT_DOTA_PlayerResource", "m_hSelectedHero")
        hero_ehandle = player_resource[hero_ehandle_key]
        hero_dt = world.recv_tables[world.classes[hero_ehandle]].dt
        hero_name = hero_dt.replace("DT_DOTA_Unit_Hero_", "").replace("_", " ")
        print hero_name
    break
```

Using Tarrasque, this could be written as

```
import tarrasque

replay = tarrasque.StreamBinding.from_file("demo.dem")
for player in replay.players:
    print player.name
    print player.hero.name
```

The code speaks for itself. Tarrasque makes it simple, easy and even fun to analyse Dota 2 replays.

1.1 Tarrasque concepts for people who know what an ehandle is

Tarrasque is a mapper between Dota2 entities (DT classes) and Python classes. Every Tarrasque class that represents an entity has a `dt_key` property that specifies the DT class that it represents, and once instantiated, every Tarrasque class has a `ehandle` property that is used to get information from the world. The current world can be accessed via `world`, and the results of `world.find(self.ehandle)` via `properties`. All this and more is documented on `DotaEntity`.

1.2 Tarrasque concepts for people who don't know what an ehandle is

Think of Tarrasque as an ORM for Dota2, except the models are already maintained, and you don't have to worry about the database. You don't have to mess about writing code to deal with the (disgusting) stuff that Dota2 does in its replays, as Tarrasque exposes the data to you in a manner that follows Python conventions; you'll get a `None` object instead of -1, and the string `"radiant"` instead of the integer 2 (where appropriate. Tarrasque understands that values have special meanings only in specific contexts). This allows you to just use the data, and not need to worry about the stuff underneath.

The one major difference between a database ORM and Tarrasque is that while most ORM models are statefull (that is, when the database changes, the model stays the same until reloaded), Tarrasque models contain no state, other than that which is needed to uniquely identify the instance (and now you know what an ehandle is). This means that you never have to do `hero.update(tick_number)` or similar; all that is handled automatically via the `StreamBinding`/ `DotaEntity` abstraction.

CHAPTER 2

Guides

2.1 Analysing game end states

One of the most common ways to get information about a game is to look at the state of the game when the ancient has died. Finding that time can be a fairly annoying process, but Tarrasque makes it quite easy. This example moves to the final tick of the game and then prints out statistics for the players:

```
import tarrasque

replay = tarrasque.StreamBinding.from_file("demo.dem", start_tick="postgame")

for player in replay.players:
    print "{} - Gold: {} - KDA: {}/{}/{}".format(player.name,
              player.earned_gold, player.kills, player.deaths. player.assists)
```

The instruction to move to the end of the replay is in the `start_tick` argument to `StreamBinding.from_file`. By saying we want to start at the `"postgame"` tick, we instruct Tarrasque to 1) locate the tick where the ancient was destroyed, and 2) move to it.

One thing to note is that while you may want to use the `GameInfo.game_time` attribute to calculate the GPM of a hero, you should first subtract 90 (1 * 60 + 30) from that value, as while the Dota2 ingame clock counts from the time the creeps spawn, the replay attribute starts 1 minute 30 seconds earlier. To calculate GPM, you might use something like this:

```
import tarrasque

replay = tarrasque.StreamBinding.from_file("demo.dem",
                     start_tick="postgame")

for player in replay.players:
    gpm = player.earned_gold * 60 / (replay.info.game_time - 90)
    print "{} - GPM: {}".format(player.name, gpm)
```

Note also that we multiply by 60, as `GameInfo.game_time` is in seconds, not minutes.

API

3.1 Stream Binding

class `StreamBinding`(*demo*, *start_tick=None*, *start_time=None*)

The StreamBinding class is Tarrasque's metaphor for the replay. Every Tarrasque entity class has a reference to an instance of this class, and when the tick of the instance changes, the data returned by those classes changes. This makes it easy to handle complex object graphs without explicitly needing to pass the Skadi demo object around.

Note: Where methods on this class take absolute tick values (i.e. the `start` and `end` arguments to `iter_ticks()`), special string arguments may be passed. These are:

- `"start"` - The start of the replay
- `"draft"` - The start of the draft
- `"pregame"` - The end of the draft phase
- `"game"` - The time when the game clock hits 0
- `"postgame"` - The time the ancient is destroyed
- `"end"` - The last tick in the replay

These values will not be 100% accurate, but should be good +-50 ticks

`buildings`

The `BuildingManager` object for the replay.

`creeps`

The `CreepManager` object for the replay.

`demo`

The Skadi demo object that the binding is reading from.

static `from_file`(*filename*, **args*, ***kwargs*)

Loads the demo from the filename, and then initialises the `StreamBinding` with it, along with any other passed arguments.

`game_events`

The game events in the current tick.

go_to_state_change (*state*)
Moves to the time when the `GameInfo.game_state` changed to the given state. Valid values are equal to the possible values of **:att:‘~GameInfo.game_state‘**, along with `"start"` and `"end"` which signify the first and last tick in the replay, respectively.

Returns the tick moved to.

go_to_tick (*tick*)
Moves to the given tick, or the nearest tick after it. Returns the tick moved to.

go_to_time (*time*)
Moves to the tick with the given game time. Could potentially overshoot, but not by too much. Will not undershoot.

Returns the tick it has moved to.

info
The `GameInfo` object for the replay.

iter_full_ticks (*start=None*, *end=None*)
A generator that iterates through the demo's 'full ticks'; sync points that occur once a minute. Should be _much_ faster than **:method:‘iter_ticks‘**.

The `start` argument may take the same range of values as the `start` argument of **:method:‘iter_ticks‘**. The first full tick yielded will be the next full tick after the position obtained via *self.go_to_tick(start)*.

The end tick may either be a tick value or a game state. The last full

tick yielded will be the first full tick after the tick value/game state change.

iter_ticks (*start=None*, *end=None*, *step=1*)
A generator that iterates through the demo's ticks and updates the `StreamBinding` to that tick. Yields the current tick.

The start parameter defines the tick to iterate from, and if not set, the current tick will be used instead.

The end parameter defines the point to stop iterating; if not set, the iteration will continue until the end of the replay.

The step parameter is the number of ticks to consume before yielding the tick; the default of one means that every tick will be yielded. Do not assume that the step is precise; the gap between two ticks will always be larger than the step, but usually not equal to it.

modifiers
The Skadi modifiers object for the tick.

players
A list of `Player` objects, one for each player in the game. This excludes spectators and other non-hero-controlling players.

prologue
The prologue of the replay.

string_tables
The string_table provided by Skadi.

tick
The current tick.

user_messages
The user messages for the current tick.

world
The Skadi wold object for the current tick.

3.2 Creep Manager

class **`CreepManager`**(*stream_binding*)

A general object that allows the user to access the creeps in the game.

`couriers`

Returns all couriers on the map

`lane`

Returns all the living lane creeps on the map.

`neutrals`

Returns all the living neutral creeps on the map.

3.3 Dota Entity

class **`DotaEntity`**(*stream_binding*, *ehandle*)

A base class for all Tarrasque entity classes.

If you plan to manually initialise this class or any class inheriting from it (and I strongly recommend against it), pass initialisation arguments by name.

`ehandle`

The ehandle of the entity. Used to identify the entity across ticks.

`exists`

True if the ehandle exists in the current tick's world. Examples of this not being true are when a `Hero` entity that represents an illusion is killed, or at the start of a game when not all heroes have been chosen.

classmethod **`get_all`**(*binding*)

This method uses the class's `dt_key` attribute to find all instances of the class in the stream binding's current tick, and then initialise them and return them as a list.

While this method seems easy enough to use, prefer other methods where possible. For example, using this function to find all `Player` instances will return 11 or more players, instead of the usual 10, where as `StreamBinding.players` returns the standard (and correct) 10.

`modifiers`

A list of the entitiy's modifiers. While this does not make sense on some entities, as modifiers can be associated with any entity, this is implemented here.

`name`

The name of an entity. This will either be equal to the `DotaEntity.raw_name` or be overridden to be a name an end user might be more familiar with. For example, if `raw_name` is `"dt_dota_nevermore"`, this value might be set to `"Nevermore"` or `"Shadow Field"`.

`owner`

The "owner" of the entity. For example, a :class:`BaseAbility` the hero that has that ability as its owner.

`properties`

Return the data associated with the handle for the current tick.

`raw_name`

The raw name of the entity. Not very useful on its own.

`stream_binding`

The `StreamBinding` object that the entity is bound to. The source of all information in a Tarrasque entity class.

team
The team that the entity is on. Options are

- `"radiant"`
- `"dire"`

tick
The current tick number.

world
The world object for the current tick. Accessed via :attr:`stream_binding`.

create_entity(*ehandle*, *stream_binding*)
Finds the correct class for the ehandle and initialises it.

find_entity_class(*dt_name*)
Returns the class that should be used to represent the ehandle with the given dt name.

register_entity(*dt_name*)
Register a class that Tarrasque will use to represent dota entities with the given DT key. This class decorator automatically sets the :attr:`~DotaEntity.dt_key` attribute.

register_entity_wildcard(*regexp*)
Similar to `register_entity`, will register a class, but instead of specifying a specific DT, use a regular expression to specify a range of DTs. For example, `Hero` uses this to supply a model for all heroes, i.e.:

```
from tarrasque.entity import *

@register_entity_wildcard("DT_DOTA_Unit_Hero_(.*)")
class Hero(DotaEntity):
    def __new__(cls, *args, **kwargs):
        # Use __new__ to dynamically generate individual hero classes
        # See tarrasque/hero.py for actual implementation
        return cls(*args, **kwargs)
```

A wildcard registration will not override a specific DT registration via `register_entity`.

3.4 Player

class **Player**(*stream_binding*, *ehandle*)
Inherits from `DotaEntity`.

Represents a player in the game. This can be a player who is controlling a hero, or a “player” that is spectating.

assists
The number of assists the player has.

buyback_cooldown_time
The game time that the buyback will come off cooldown. If this is 0, the player has not bought back.

deaths
The number of times the player has died.

denies
The number of denies on creeps that the player has.

earned_gold
The total earned gold by the user. This is not net worth; it should be used to calculate gpm and stuff.

`has_buyback`
Can the player buyback (regardless of their being alive or dead).

`hero`
The `Hero` that the player is playing in the tick. May be `None` if the player has yet to choose a hero. May change when the `game_state` is `"pre_game"`, due to players swapping their heroes.

`index`
The index of the player in the game. i.e. 0 is the first player on the radiant team, 9 is the last on the dire

This is `None` for the undefined player, which should be ignored.

`kills`
The number of times the player has killed an enemy hero.

`last_buyback_time`
The `game_time` that the player bought back.

`last_hits`
The number of last hits on creeps that the player has.

`name`
The Steam name of the player, at the time of the game being played.

`reliable_gold`
The player's reliable gold.

`steam_id`
The Steam ID of the player.

`streak`
The current kill-streak the player is on

`team`
The player's team. Possible values are

- `"radiant"`
- `"dire"`
- `"spectator"`

`total_gold`
The sum of the player's reliable and unreliable gold.

`unreliable_gold`
The player's unreliable gold.

3.5 Game Info

class `GameInfo`(*stream_binding*, *ehandle*)
Inherits from `DotaEntity`

The GameInfo contains the macro state of the game; the stage of the game that the tick is in, whether the tick is in day or night, the length of the game, etc etc.

`active_team`
The team that is currently banning/picking.

`banned_heroes`
List of currently banned heroes. 0-4 are radiant picks, 5-9 dire. Bans that have not yet been done have value None.

`captain_ids`
IDs of the picking players (captains)

`draft_start_time`
The time that the game_state changed to `draft`.

`extra_time`
Extra time left for both teams. Index 0 is radiant, index 1 is dire

`game_end_time`
The time that the game_state changed to `postgame`.

`game_mode`
The mode of the dota game. Possible values are:

- `"none"`
- `"all pick"`
- `"captain's mode"`
- `"random draft"`
- `"single draft"`
- `"all random"`
- `"intro"`
- `"diretide"`
- `"reverse captain's mode"`
- `"greeviling"`
- `"tutorial"`
- `"mid only"`
- `"least played"`
- `"new player pool"`
- `"compendium matchmaking"`

`game_start_time`
The time that the game_state changed to `game`.

`game_state`
The state of the game. Potential values are:

- `"loading"` - Players are loading into the game
- `"draft"` - The draft state has begun
- `"strategy"` - Unknown
- `"pregame"` - The game has started but creeps have not been spawned
- `"game"` - The main game, between the first creep spawn and the ancient being destroyed
- `"postgame"` - After the ancient has been destroyed
- `"disconnect"` - Unknown

`game_time`
The time in seconds of the current tick.

`game_winner`
The winner of the game.

`load_time`
The time that the game_state changed to `loading`.

`match_id`
The unique match id, used by the Steam API and stuff (i.e. DotaBUff and friends).

`pausing_team`
The team that is currently pausing. Will be `None` if the game is not paused, otherwise either `"radiant"` or `"dire"`.

`pick_state`
The current pick/ban that is happening. `None` if no pick or ban is happening. If the `game_mode` is not `"captain's mode"`, the possible values are:

- `"all pick"`
- `"single draft"`
- `"random draft"`
- `"all random"`

Otherwise, the current pick and ban is returned in a tuple of the type of draft action and the index. For example, if the current tick was during the 5th ban of a captains mode game, the value of `pick_state` would be `("ban", 5)`. `active_team` could then be used to work out who is banning. Alternatively, if it was the 2nd pick of the game, it would be `("pick", 2)`.

`pregame_start_time`
The time that the game_state changed to `pregame`.

`replay_length`
The length in seconds of the replay.

`selected_heroes`
List of currently picked heroes. 0-4 are radiant picks, 5-9 dire. Picks that have not yet been done have value None.

`starting_team`
The team that begins the draft.

3.6 Ability

class `BaseAbility`(*stream_binding*, *ehandle*)
Base class for all abilities. Currently does not delegate to other classes, but can do so.

`cast_range`
The distance from the hero's position that this spell can be cast/targeted at.

`cooldown_length`
How long the goes on cooldown for every time it is cast.

`is_on_cooldown`
Uses `off_cooldown_time` and `GameInfo.game_time` to calculate if the ability is on cooldown or not.

`is_ultimate`
Use's the abilities position in `Hero.abilities` to figure out if this is the ultimate ability.

TODO: Check this is reliable

level
The number of times the ability has been leveled up.

mana_cost
The mana cost of the spell

off_cooldown_time
The time the ability comes off cooldown. Note that this does not reset once that time has been passed.

3.7 Base NPC

class BaseNPC (*stream_binding*, *ehandle*)
A base class for all NPCs, even ones controllable by players.

abilities
A list of the NPC's abilities.

health
The NPC's current HP.

health_regen
The NPC's health regen per second.

inventory
A list of the NPC's items.

is_alive
A boolean to test if the NPC is alive or not.

level
The NPC's level. See `Hero.ability_points` for unspent level up ability points.

life_state
The state of the NPC's life (unsurprisingly). Possible values are:

- `"alive"` - The hero is alive
- `"dying"` - The hero is in their death animation
- `"dead"` - The hero is dead
- `"respawnable"` - The hero can be respawned
- `"discardbody"` - The hero's body can be discarded

`"respawnable"` and `"discardbody"` shouldn't occur in a Dota2 replay

mana
The NPC's current mana.

mana_regen
The NPC's mana regen per second.

max_health
The NPC's maximum HP.

max_mana
The NPC's maximum mana.

position
The (x, y) position of the NPC in Dota2 map coordinates

3.8 Hero

While each hero has a distinct class, not all have classes that are defined in source code. This is because the `Hero` class registers itself as a wildcard on the DT regexp `"DT_DOTA_Unit_Hero_*"`, and then dynamically generates hero classes from the ehandle. The generated classes simply inherit from the `Hero` and have different values for `dt_key` and `name`.

class `Hero` (*stream_binding*, *ehandle*)
While all hero classes inherit from this class, it is unlikely that this class will ever need to be instantiated.

`ability_points`
Seems to be the number of ability points the player can assign.

`agility`
The hero's agility (from levels, items, and the attribute bonus).

static `get_all_heroes` (*stream_binding*)

Overrides DotaEntity.get_all in order to return all heroes with the prefix
`` `"DT_DOTA_Unit_Hero"` ``, as there is never any results for

`` `"DT_DOTA_BaseNPC_Hero"` ``, and it also wouldn't be of any use to devs.

`intelligence`
The hero's intelligence (from levels, items, and the attribute bonus).

`name` = None
The name of the hero. For the base `Hero` class, this is `None`, but it is set when a subclass is created in the __new__ method.

`natural_agility`
The hero's agility from levels.

`natural_intelligence`
The hero's intelligence from levels.

`natural_strength`
The hero's strength from levels.

`player`
The player that is playing the hero.

`recent_damage`
The damage taken by the hero recently. The exact time period that classifies as "recently" is around 2/3 seconds.

TODO: Find exact value

`replicating_hero`
The `Hero` the current hero is "replicating" **[#f1]_**. If the instance is not an illusion (which use the `Hero` class also), this will be `None`. There is no guarantee that that this hero will exist (see `DotaEntity.exists`) if the hero is someone like Phantom Lancer, who may have an illusion which creates other illusions, and then dies. However, this is still a useful property for tracking illusion creation chains

`respawn_time`
Appears to be the absolute time that the hero respawns. See `game_time` for the current time of the tick to compare.

TODO: Check this on IRC

`spawned_at`
The time (in `game_time` units) the hero spawned at.

TODO: Check this in game.

`strength`
The hero's strength (from levels, items, and the attribute bonus).

`xp`
The hero's experience.

3.9 Game Events

class **`GameEvent`** (*stream_binding*, *name*, *properties*)
Base class for all game events. Handles humanise and related things.

`name` **= None**
The name of the GameEvent. i.e. `"dota_combatlog"`, `"dota_chase_hero"`.

`create_game_event` (*stream_binding*, *data*)
Creates a new GameEvent object from a stream binding and the un-humanized game event data.

`find_game_event_class` (*event_name*)
Given the name of an event, finds the class that should be used to represent it.

`register_event` (*event_name*)
Register a class as the handler for a given event.

`register_event_wildcard` (*event_pattern*)
Same as `register_event()` but uses a regex pattern to match, instead of a static game event name.

3.10 Combat Log

class **`CombatLogMessage`** (*stream_binding*, *name*, *properties*)
A message in the combat log.

`attacker_name`
The name of the attacker in the event.

`health`
The health of the unit being attacked, for 'heal' and 'damage' events.

`inflictorname`
The name of the "inflictor" (wtf is that?). Used to id modifiers.

`source_name`
The name of the source of the event.

`target_name`
The name of the entity that was targeted in the event. Note that this is not the dt name or "pretty" name, this is the `DotaEntity.raw_name`. So for a message where Shadow Field is being attacked, this would be `"dt_dota_nevermore"`.

`timestamp`
The timestamp this combat log message corresponds to.

`type`
The type of event this message signifies. Options are:

- `"damage"` - One entity is damaging another
- `"heal"` - One entity is healing another
- `"modifier added"` - A modifier is being added to an entity
- `"modifier removed"` - A modifier is being removed from an entity
- `"death"` - An entity has died.

value
The value of the event. Can have various different meanings depending on the `type`.

3.11 Item

class Item(*stream_binding*, *ehandle*)
Item class

alertable
Presumably whether you can right-click 'Alert allies' with it (ex: Smoke, Arcane Boots, 'Gather for Arcane Boots here!')

cooldown_length
These are all the same as the functions in the ability class, I'm lazy, go read them, they are fairly self-explanatory :D

current_charges
Presumably the item's current charges (ex: 7 for Diffusal if used once)

disassemblable
Presumably whether you can disassemble the item (ex: Arcane Boots)

droppable
Presumably if the item is droppable (ex: not Aegis)

initial_charges
Presumably charges when item is bought (ex: 8 for diffusal)

killable
Presumably whether the item can be denied (ex: not Gem)

off_cooldown_time
The time when the item will come off cooldown

permanent
Seems to be if the item will disappear when it runs out of stacks (i.e consumable. Ex: Tango, not Diffusal)

purchasable
Presumably whether you can buy the item or not (ex: not Aegis)

purchase_time
The time when the item was purchased

purchaser
The hero object of the purchaser of the item

recipe
Presumably whether the item is a recipe or not (ex: any Recipe)

requires_charges
Presumably whether the item needs charges to work (ex: Diffusal)

`sellable`
Presumably whether the item can be sold or not (ex: Not BKB)

`sharability`
Presumably whether the item can be shared (ex: Tango, RoH)

`stackable`
Presumably whether the item can be stacked (ex: Wards)

CHAPTER 4

Indices and tables

- *genindex*
- *search*

Python Module Index

t

tarrasque.ability, 13
tarrasque.basenpc, 14
tarrasque.binding, 7
tarrasque.combatlog, 16
tarrasque.creeps.manager, 9
tarrasque.entity, 9
tarrasque.gameevents, 16
tarrasque.gameinfo, 11
tarrasque.hero, 15
tarrasque.item, 17
tarrasque.player, 10