INF8808 - Projet Starcraft II

Philippe Carphin

March 13, 2019

Contents

1 INF8808 - Projet Starcraft II

1.1 The sc2replay module and how starcraft 2 works

1.1.1 Overview

The python module is `sc2reader` and it is available with pip. It offers the function `sc2reader.load_replay(filename)` which returns an python object.

This object has a bunch of attributes, most notable of which is the `game_events` attribute. Starcraft 2 is a deterministic state machine. A game is recorded by recording all these game events, and the game is replayed by having the state machine consume these events.

1.1.2 An individual event

The types of actual events are stored in `sc2reader.events` and in `sc2reader.events.game` there is a subset of these types. These are the types that are interesting.

It is useful to create a replay, get all the events of a certain type, and then put a breakpoint in your IDE to inspect the event objects for it's attributes.

Knowing what attributes a certain type of event has allows us to design a selector to select events matching certain criteria. We can then use the `select_events` method of the wrapper to get the selected events.

```
python3 -m bpython
bpython version 0.17.1 on top of Python 3.7.2 /usr/local/bin/python3
>>> import sc2reader
>>> sc2reader.events.

 AddToControlGroupEvent            BasicCommandEvent
 CameraEvent                       ChatEvent
 CommandEvent                      ControlGroupEvent
 DataCommandEvent                  Event
 GameEvent                         GameStartEvent
 GetControlGroupEvent              HijackReplayGameEvent
 Length                            MessageEvent
 PingEvent                         PlayerLeaveEvent
 PlayerSetupEvent                  PlayerStatsEvent
 ProgressEvent                     ResourceRequestCancelEvent
 ResourceRequestEvent              ResourceRequestFulfillEvent
 ResourceTradeEvent                SelectionEvent
 SetControlGroupEvent              TargetPointCommandEvent
 TargetUnitCommandEvent            TrackerEvent
 UnitBornEvent                     UnitDiedEvent
 UnitDoneEvent                     UnitInitEvent
 UnitOwnerChangeEvent              UnitPositionsEvent
 UnitTypeChangeEvent               UpdateTargetPointCommandEvent
 UpdateTargetUnitCommandEvent      UpgradeCompleteEvent
 UserOptionsEvent                  absolute_import
 base                              chain
 clamp                             create_command_event
 create_control_group_event        division
 functools                         game
 loggable                          message
 print_function                    tracker
 unicode_literals


python3 -m bpython
bpython version 0.17.1 on top of Python 3.7.2 /usr/local/bin/python3
>>> import sc2reader
>>> sc2reader.events.game.

 AddToControlGroupEvent           BasicCommandEvent
```

```
CameraEvent
CommandEvent
ControlGroupEvent
DataCommandEvent
Event
GameEvent
GameStartEvent
GetControlGroupEvent
HijackReplayGameEvent
Length
PlayerLeaveEvent
ResourceRequestCancelEvent
ResourceRequestEvent
ResourceRequestFulfillEvent
ResourceTradeEvent
SelectionEvent
SetControlGroupEvent
TargetPointCommandEvent
TargetUnitCommandEvent
UpdateTargetPointCommandEvent
UpdateTargetUnitCommandEvent
UserOptionsEvent
absolute_import
chain
create_command_event
create_control_group_event
division
loggable
print_function
unicode_literals
```

Each of these events has a different set of attributes. For example, some events have a `location` attribute, some don't. `CameraEvent` has a `location` attribute but `TargetUnitCommandEvent` does not. The targeted unit will have a location and the player will be looking at the location of their last `CameraEvent`.

1.2 Wrapper interface

To allow easy data gathering, the following wrapper interface can be used:

```
import pysc2
my_replay_wrapper = pysc2.SC2ReplayWrapper('spawningtool_replays/dark_v_Solar_Game1_Por
```

We can get events by type or by any other predicate. We can categorize them using a helper function, and we can draw scatter plots with color and also bar charts (although bar charts are not going to be in our project, they might be useful to analyse various sets of data.

The best and most up-to-date examples can be found in the `test_pysc2.py` file but the following give a brief overview (I updated it after I wrote all the code, but I did write this before I wrote the code, so things might be different but the idea will stay the same.

1. Get events of a certain type:

```
def selector(e):
    return isinstance(e, sc2reader.events.game.GetControlGroupEvent)
selected_events = my_replay_wrapper.select_from_list(my_replay_wrapper._replay.gam
```

It's a bit weird that you have to pass the list of events because we're calling a method of the object that owns those lists, but it's because there are two lists that we will want to select events from, the `_replay.events` and the `_replay.game_events` lists. So for now, this is how you tell it which of the two lists to select from.

I did make convenience functions `select_from_events()` and `select_from_game_events()` to avoid that funny looking stuff.

Also, in coding, one might create a list and select from that list, and that might be useful.

2. For more precision

```
def player_2_zealot_born_selector(event):
    return (isinstance(event, sc2reader.events.UnitBornEvent):
            and event.player == 2
            and "bla bla to ceck if it's a zealot")
selected_events = my_replay_wrapper.select_from_events(player_2_zealot_born_select
```

3. For locations

   If a list of events needs locations based on where the player is looking, then we have

```
def color_map(event, i):
    if i % 2 == 0:
        return 'gray'
    else:
        return 'blue'

locations = my_replay.create_locations(event_list)

colors = list(map(event_list, color_map))
plt.scatter(x=locations[0,:], y=locations[1,:], c=colors)
plt.show()
```

```
def category_map(event):
    if ...:
        return "offensive"
    elif ...:
        return "defensive"

my_replay.category_bar_chart(category_map)

def cumulative_map(event_list, category_map, value_map):
    values = {}
    for e in event_list:
        cat = category_map(e)
        if cat is not None:
            values[cat] += value_map(e)

def value_map(event):
    return event.damage
def category_map(event):
    if ...:
        return "offensive"
    elif ...:
        return "defensive"
def player_2_damage_done_selector(event):
    return (isinstance(xyz)
            and event.pid == 2
            and event.something_else == this)

my_replay.cumulative_bar_chart(player_2_damage_done_selector, category_map, value_
```