Adrián Heras' PyCatan

Manual del desarrollador

Introducción

A lo largo del siguiente documento se detallan las clases más importantes del entorno de simulación, además de la estructura del JSON, para poder programar bots aplicables a dicho entorno. Esta documentación posee sobre las funciones de cada una de las clases mostradas información indicando los valores de entrada, de salida y una breve explicación de qué hace una función, similar a cómo se encuentra en el *pydoc*.

1 Clases del simulador

A continuación se muestran las clases más importantes que posee el entorno de simulación, así como una explicación de para qué sirven, y detalle sobre sus funciones.

1.1 Board

La clase *Board* representa una instancia del tablero. Por tanto, posee métodos y variables que hacen referencia a este. La clase *Board* posee métodos que permiten a un bot buscar nodos válidos donde poder construir pueblos, carreteras o ciudades, lo que la hacen indispensable para los bots.

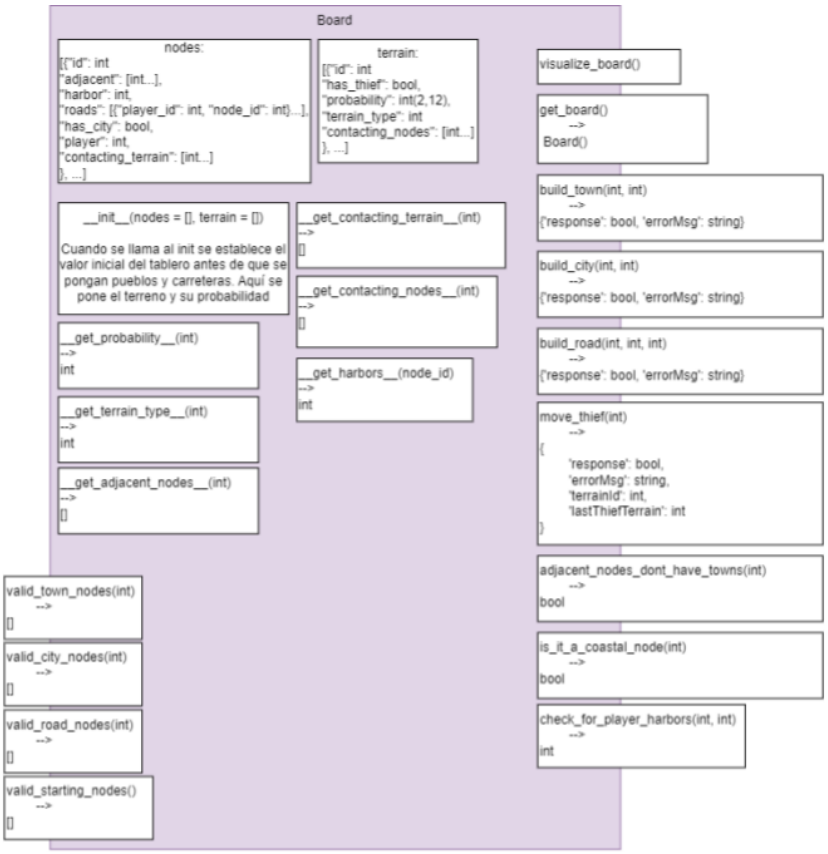

Figura 1, Clase Board

1.1.1 Variables

En este apartado se explican las variables internas que posee la clase *Board*:

La variable **nodes** posee información de todos los nodos del tablero. Se compone de un array de objetos nodo. Ésta es su estructura:

```
nodes:
[{
"id": int
"adjacent": [int...],
"harbor": int,
"roads": [{"player_id": int, "node_id": int}...],
"has_city": bool,
"player": int,
```

```
"contacting_terrain": [int...]
}, ...]
```

Dentro del objeto, se encuentran las siguientes claves:

- **id**: número que identifica el nodo en el tablero.
- **adjacent**: array con las IDs de los nodos adyacentes a éste.
- **harbor**: número que indica si este nodo posee puerto y de qué tipo en caso de poseerlo.
- **roads**: array de objetos que indican a quién le pertenece la carretera y con qué otro nodo está conectada.
- **has_city**: booleano que indica si este nodo posee una ciudad construida sobre él.
- **player**: número que indica si este nodo le pertenece a un jugador. Un nodo solo le pertenece a un jugador si tiene un pueblo o una ciudad sobre él, luego si este número es diferente a -1, implica que algún jugador ha construido un pueblo sobre él, y si *has_city* es *True* implica que hay una ciudad sobre este nodo.
- **contacting_terrain**: array con las IDs de las piezas de terreno en contacto con este nodo.

La variable **terrain** posee información sobre todas las piezas de terreno del tablero. Se compone de un array de objetos, de una manera similar a la variable *nodes*. Ésta es su estructura:

```
terrain:
[{
"id": int
"has_thief": bool,
"probability": int(2,12),
"terrain_type": int
"contacting_nodes": [int...]
}, ...]
```

Las siguientes claves son las que forman el objeto:

- **id**: número que identifica la pieza de terreno.
- **has_thief**: booleano que indica si esta pieza de terreno tiene al ladrón sobre ella y por tanto no puede generar materiales.
- **probability**: número que indica el valor necesario a sacar en el dado para que esta pieza de terreno genere materiales.
- **terrain_type**: número que indica el tipo de pieza de terreno que es.
- **contacting_nodes**: array con las IDs de los nodos en contacto con esta pieza de terreno.

1.1.2 Funciones privadas

En esta subsección se añaden las funciones privadas de la clase *Board*. Se utilizan exclusivamente en la inicialización de la clase, para generar las variables *nodes* y *terrain*.

__get_probability__: Función que devuelve la probabilidad a una pieza de terreno otorgada.

Valores de entrada:

- *terrain_id*: Número que representa la ID de una pieza de terreno.

Valores de salida:

- *int*: Valor de la probabilidad perteneciente a la ficha de terreno otorgada.

__get_terrain_type__: Función que devuelve el tipo de terreno a una pieza de terreno otorgada.

Valores de entrada:

- *terrain_id*: Número que representa la ID de una pieza de terreno.

Valores de salida:

- *int*: Valor del tipo de terreno perteneciente a la ficha de terreno otorgada.

__get_adjacent_nodes__: Función que devuelve los nodos adyacentes a uno otorgado.

Valores de entrada:

- *node_id*: Número que representa la ID de un nodo.

Valores de salida:

- *[int, …]*: Array que posee las IDs de todos los nodos adyacentes a uno otorgado.

__get_contacting_terrain__: Función que devuelve las piezas de terreno adyacentes a un nodo otorgado.

Valores de entrada:

- *node_id*: Número que representa la ID de un nodo.

Valores de salida:

- *[int, …]*: Array que posee las IDs de todas las piezas de terreno adyacentes al nodo otorgado.

__get_contacting_nodes__: Función que devuelve los nodos adyacentes a una pieza de terreno otorgada.

Valores de entrada:

- *terrain_id*: Número que representa la ID de una pieza de terreno.

Valores de salida:

- *[int, …]*: Array que posee las IDs de todos los nodos adyacentes a uno otorgado.

__get_harbors__: Función que devuelve el puerto correspondiente de un nodo. Se recomienda mirar la clave “harbor” del objeto nodo de la variable *nodes* para comprobar el puerto. Ésta se utiliza solo en la iniciación de dicha variable.

Valores de entrada:

- *node_id*: Número que representa la ID de un nodo.

Valores de salida:

- *int*: Número que representa si tiene o no puerto el nodo, y el tipo de puerto en caso de tenerlo.

1.1.3 Funciones públicas

En este apartado se detallan las funciones públicas de la clase *Board*. Entre ellas se hallan algunas que pueden resultar especialmente útiles para el desarrollo de un bot.

visualize_board: Función que muestra por pantalla todos los nodos y piezas de terreno del tablero. Se utiliza por si interesa ver el estado del tablero en cierto momento.

get_board: Función que devuelve esta instancia del tablero.

Valores de salida:

- *Board()*: instancia de la clase *Board* actual.

build_town: Función que construye un pueblo en un nodo otorgado. La función se asegura de que no tenga un pueblo ya antes y de que la construcción sea legal, es decir, que no esté adyacente a otro pueblo y haya al menos una carretera del jugador conectada al nodo.

Valores de entrada:

- *player*: Número que representa la ID del jugador.
- *node_id*: Número que representa la ID de un nodo.

Valores de salida:

- *{'response': bool, 'error_msg': string}*: Objeto de salida que confirma o niega si se ha construido un pueblo en el nodo seleccionado.

build_city: Función que construye una ciudad en un nodo otorgado. La función se asegura de que ya le pertenezca al jugador antes de crear la ciudad.

Valores de entrada:

- *player*: Número que representa la ID del jugador.
- *node_id*: Número que representa la ID de un nodo.

Valores de salida:

- *{'response': bool, 'error_msg': string}*: Objeto de salida que confirma o niega si se ha construido una ciudad en el nodo seleccionado.

build_road: Función que construye una carretera entre los nodos otorgados. Esta función comprueba que no hayan carreteras ya antes, del propio jugador o de otro, además de asegurarse de que haya al menos una carretera o un nodo del jugador que la intenta construir antes, para que la construcción sea legal.

Valores de entrada:

- *player*: Número que representa la ID del jugador
- *starting_node*: Número que representa la ID del nodo inicial.
- *ending_node*: Número que representa la ID del nodo final

Valores de salida:

- *{'response': bool, 'error_msg': string}*: Objeto de salida que confirma o niega si se ha construido una carretera en el par de nodos otorgado.

move_thief: Permite mover al ladrón a la pieza de terreno especificada.

Valores de entrada:

- *terrain*: Número que representa la ID de una pieza de terreno

Valores de salida:

- *{'response': bool, 'error_msg': string, 'terrain_id': int, 'last_thief_terrain': int}*: Objeto de salida que confirma o niega si se ha movido al ladrón, en qué pieza de terreno se encuentra ahora y en qué pieza de terreno se encontraba antes, para la correcta visualización en el visualizador.

adjacent_nodes_dont_have_towns: Comprueba si los nodos adyacentes no tienen un pueblo, para asegurarse a la hora de construir uno de que el nodo es legal.

Valores de entrada:

- *node_id*: Número que representa la ID de un nodo.

Valores de salida:

- *bool*: Devuelve True si no hay pueblos adyacentes, False si hay al menos uno.

is_it_a_coastal_node: Comprueba si el nodo otorgado es uno costero. Utilizado principalmente durante el inicio de la partida, donde es ilegal colocar el primer pueblo en la costa de la isla.

Valores de entrada:

- *node_id*: Número que representa la ID de un nodo.

Valores de salida:

- *bool*: Devuelve True si es un nodo costero, False si no lo es.

check_for_player_harbors: Comprueba si el jugador posee un puerto del tipo especificado. Si no se especifica tipo, comprueba si tiene un puerto 3 a 1. Esta función solo se utiliza en el GameDirector para comprobar si un jugador posee un puerto de un material específico en caso de querer comerciar con la banca.

Valores de entrada:

- *player*: Número que representa la ID del jugador.
- *material_harbor* (opcional): Número que representa la ID del tipo de material del que se desea buscar el puerto.

Valores de salida:

- *int*: Número que representa el tipo de puerto del jugador.

valid_town_nodes: Devuelve un array con todos los nodos viables para construir un pueblo, dada la ID de un jugador.

Valores de entrada:

- *player_id*: Número que representa la ID del jugador.

Valores de salida:

- *[int,...]*: Array de IDs de los nodos donde el jugador puede construir un pueblo.

valid_city_nodes: Devuelve un array con todos los nodos viables para construir una ciudad, dada la ID de un jugador.

Valores de entrada:

- *player_id*: Número que representa la ID del jugador.

Valores de salida:

- *[int,...]*: Array de IDs de los nodos donde el jugador puede construir una ciudad.

valid_road_nodes: Devuelve un array de objetos con todas las carreteras posibles para construir, dada la ID de un jugador.

Valores de entrada:

- *player_id*: Número que representa la ID del jugador.

Valores de salida:

- *[{'starting_node': int, 'finishing_node': int}, …]*: Array objetos con todas las carreteras que el jugador tiene permitido construir.

1.2 BotInterface

La clase *BotInterface* es la interfaz de desarrollo del bot. Con esta interfaz se han creado los bots de ejemplos que vienen junto al proyecto: RandomBot y AdrianHerasBot.

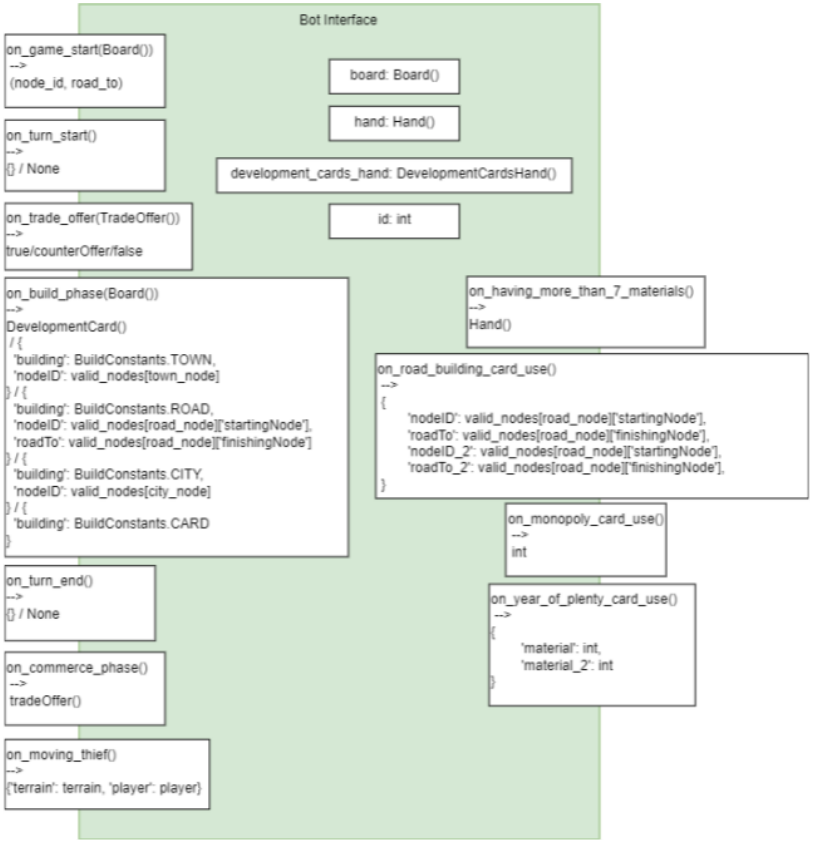

Figura 2, Clase BotInterface

1.2.1 Variables

La clase BotInterface posee múltiples variables que se definen al instanciarla. Todas llevan datos necesarios del bot, que aunque el bot cambie no afecta a la copia de estas variables que posee el GameManager. La variables son las siguientes:

- **board**: la variable *board* es una instancia del tablero. Durante el trigger *on_build_phase* se le pasará la instancia del tablero actualizada al bot, para que los cálculos de construcción puedan ser correctos.
- **hand**: la variable *hand* se encarga de llevar la cuenta de los materiales que tiene el bot en mano. La variable *hand* es modificada por el *GameManager* cuando se entregan materiales a los jugadores al inicio del turno.
- **development_cards_hand**: la variable *development_cards_hand* posee la mano de cartas de desarrollo del bot. Cuando un bot construye una carta de desarrollo, esta variable es modificada por el *GameManager* para añadirle la carta que ha construido.
- **id**: la variable *id* muestra la ID del jugador del bot. Ésta sirve para saber qué jugador eres y poder llamar a las funciones de la clase *Board* sin problemas.

1.2.2 Funciones públicas

Las funciones que la interfaz pide implementar son triggers que el *GameManager* llama. En caso de dejar las funciones sin determinar, están preparadas para devolver “None”, lo que el *GameManager* interpreta como que el boy pasa turno, a menos que sea necesaria una

respuesta, en cuyo caso la elige el *GameManager* por el bot.

on_game_start: Trigger que se llama al inicio de la partida dos veces. Permite elegir dónde poner un pueblo y una carretera para empezar la partida.

Valores de entrada:

- *board_instance*: Instancia de la clase *Board* para que el bot pueda ver donde han colocado sus pueblos el resto de jugadores.

Valores de salida:

- *int, int*: Tupla con los datos del nodo donde se coloca el pueblo, y el nodo al que apunta la carretera.

on_turn_start: Trigger que se llama al inicio del turno del jugador. Sirve exclusivamente para jugar cartas de desarrollo.

Valores de salida:

- *None / DevelopmentCard*: Se devuelve None en caso de querer terminar la fase o una carta de desarrollo de la mano en caso de querer jugar dicha carta.

on_trade_offer: Trigger que se llama cuando llega un intercambio propuesto por otro bot.

Valores de entrada:

- *incoming_trade_offer*: Instancia de la clase *TradeOffer* con la oferta de intercambio de materiales.

Valores de salida:

- *True / TradeOffer / False / None*: Se devuelve True si se quiere aceptar la oferta. None o False si se quiere directamente negar la oferta. Y se devuelve una instancia de *TradeOffer* si se quiere empezar una contraoferta

on_build_phase: Trigger que se llama cuando comienza la fase de construcción. Permite construir carreteras, pueblos, ciudades y cartas de desarrollo. También se permiten jugar cartas de desarrollo.

Valores de entrada:

- *board_instance*: Instancia de la clase *Board* para que el bot pueda ver el tablero actualizado antes de construir.

Valores de salida:

- *None / DevelopmentCard / {'building': string} / {'building': string, 'node_id': int} / {'building': string, 'node_id': int, 'road_to': int}*: Se devuelve None si no se quiere continuar con la fase de construcción. Se devuelve una carta de desarrollo si se quiere jugar una carta. Se devuelve un objeto si se quiere construir. El objeto se envía sin claves "node_id" y "road_to" en caso de ser una carta de desarrollo. El objeto posee "node_id" si es un pueblo o una ciudad. El objeto posee "road_to", además de "node_id", en caso de ser una carretera.

on_turn_end: Trigger que se llama cuando termina el turno del jugador. Sirve exclusivamente para jugar cartas de desarrollo.

Valores de salida:

- *None / DevelopmentCard*: Se devuelve None en caso de querer terminar la fase o una carta de desarrollo de la mano en caso de querer jugar dicha carta.

on_commerce_phase: Trigger que se llama durante la fase de comercio del turno. Permite hacer ofertas a otros jugadores o con la banca. También se permiten jugar cartas de

desarrollo.

Valores de salida:

- *{'gives': int, 'receives': int} / TradeOffer*: Se devuelve un objeto en caso de querer comerciar con la banca. Se devuelve una instancia de *TradeOffer* en caso de querer enviar una petición de comercio a todos los demás bots.

on_moving_thief: Trigger que se llama cuando a este bot le toca mover al ladrón a otra ficha de terreno. Permite elegir a qué ficha de terreno moverlo y a qué jugador adyacente a dicha ficha se le quiere robar un material.

Valores de salida:

- *None / {'terrain': int, 'player': int}*: En caso de devolver None, el *GameManager* moverá al ladrón a su casilla inicial y no robará a ningún jugador. En caso de devolver el objeto, éste debe poseer una clave "terrain" con la ID de la ficha de terreno donde se quiere colocar al ladrón y una clave "player" con la ID del jugador.

on_having_more_than_7_materials: Trigger que se lanza cuando se poseen más de 7 materiales en mano y ha salido un 7 en el dado. Deben descartarse la mitad de las cartas que se posee, redondeado hacia abajo. En caso de que no se hayan descartado todas, el *GameManager* las descartará aleatoriamente.

Valores de salida:

- *Hand*: Se debe devolver la instancia de la clase *Hand* del jugador para que pueda hacer la comprobación de los materiales descartados.

on_road_building_card_use: Trigger lanzado cuando el bot juega una carta de construcción de carreteras. Permite elegir qué dos carreteras se quieren construir.

Valores de salida:

- *None / {'node_id': int, 'road_to': int, 'node_id_2': None / int, 'road_to_2': None / int}*: En caso de devolver None no se construirá ninguna carretera. Si solo es posible construir una carretera, las claves "node_id_2" y "road_to_2" deben ser None. En caso de construir dos carreteras, se escriben en las claves "node_id" y "node_id_2" las IDs de los nodos iniciales y en "road_to" y "road_to_2" los nodos finales de las carreteras.

on_monopoly_card_use: Trigger lanzado cuando el bot juega una carta de monopolio. Se elige de qué material se desea poseer el monopolio.

Valores de salida:

- *None / int*: En caso de devolver None no se roba ningún material a ningún rival y se malgasta la carta. En caso de devolver la ID de un material, se le roba todas las cartas de ese material a todos los oponentes.

on_year_of_plenty_use: Trigger lanzado cuando el bot juega una carta de año de la cosecha. Se eligen 2 materiales, que pueden ser el mismo dos veces, y se obtiene un material de cada tipo.

Valores de salida:

- *None / {'material': int, 'material_2' int}*: En caso de devolver None no se obtienen materiales y la carta es malgastada. En caso de devolver un objeto, es necesario poner ambas claves con la ID de los materiales que se desean obtener. Ambos materiales pueden ser el mismo.

2 Estructura del JSON

En este apartado se muestra la estructura del fichero JSON extraído por el entorno de visualización entre partidas. El fichero se compone de un array de objetos partida. Cada partida se divide en dos bloques: *setup* y *game*. En el apartado de *setup* se hallan todos los datos de preparación de partida, como el tablero o la posición inicial de los pueblos y carreteras. En el apartado de *game* se encuentran todos los datos de la partida, cada una de las rondas, con cada turno y cada fase de cada jugador. La estructura del archivo es la siguiente:

```
[
  {
    "setup": {
       "board": {
           "board_nodes": [
                {
                    "id": int,
                    "adjacent": [int, ...],
                    "harbor": int,
                    "roads": [
                        {
                            "player_id": int
                            "node_id": int
                        }, ...
                    ],
                    "has_city": bool,
                    "player": int,
                    "contacting_terrain": [int, ...]
                }, ...
           ],
           "board_terrain": [
               {
                    "id": int,
                    "has_thief": bool,
                    "probability": int,
                    "terrain_type": int,
                    "contacting_nodes": [int, ...]
               }, ...
           ]
       },
       "P0": [
          {
             "id": int,
             "road": int,
          },
          {
             "id": int,
             "road": int,
          }
       ],
       "P1": [...],
       "P2": [...],
       "P3": [...]
    },
    "game": {
        "round_0": {
            "turn_P0"; {
                "start_turn": {
                    "development_card_played": [
```

```
                        {
                            objeto variable
                        }
                    ],
                    "dice": int,
                    "actual_player": int,
                    "hand_P0": {
                        "cereal": int,
                        "mineral": int,
                        "clay": int,
                        "wood": int,
                        "wool": int
                    },
                    "total_P0": int,
                    "hand_P1": {...},
                    "total_P1": int,
                    "hand_P2": {...},
                    "total_P2": int,
                    "hand_P3": {...},
                    "total_P3": int,
                },
                "commerce_phase": [
                    {
                        objeto variable
                    },
                ],
                "build_phase": [
                    {
                        objeto variable
                    }
                ],
                "end_phase": {
                    "development_card_played": [
                        {
                            objeto variable
                        }
                    ],
                    "victory_points": {
                        "J0": int,
                        "J1": int,
                        "J2": int,
                        "J3": int
                    }
                }
            },
            "turn_P1"; {...},
            "turn_P2"; {...},
            "turn_P3"; {...},
        }, ...
    }
  }, ...
]
```

Dentro de esta estructura se observan múltiples “objeto variable”. Esto se debe a que los objetos de esas claves varían en función de uno de los valores internos. A continuación se detallan todas las posibilidades:

commerce_phase: Objeto que muestra la fase de comercio. La fase de comercio es un array compuesto por diferentes objetos que cambian en función de lo que haya hecho el bot

durante la fase. Los posibles objetos son los siguientes:

En caso de haber una oferta de intercambio para los demás jugadores, el objeto del array de *commerce_phase* tendrá esta forma. Los apartados *giver* y *receiver* reciben un entero que representa la ID del jugador.

```
{
    "trade_offer": {
        "gives":{
            "cereal": int,
            "mineral": int,
            "clay": int,
            "wood": int,
            "wool": int
        },
        "receives":{
            "cereal": int,
            "mineral": int,
            "clay": int,
            "wood": int,
            "wool": int
        }
    },
    "harbor_trade": bool,
    "inviable": bool,
    "answers": [
        [
            {
                "count": int,
                "trade_offer": {
                        "gives":{
                        "cereal": int,
                        "mineral": int,
                        "clay": int,
                        "wood": int,
                        "wool": int
                    },
                    "receives":{
                        "cereal": int,
                        "mineral": int,
                        "clay": int,
                        "wood": int,
                        "wool": int
                    }
                },
                "giver": int,
                "receiver": int,
                "response": bool / {
                    "gives": {
                        "cereal": int,
                        "mineral": int,
                        "clay": int,
                        "wood": int,
                        "wool": int
                    },
                    "receives": {
                        "cereal": int,
                        "mineral": int,
                        "clay": int,
                        "wood": int,
```

```
                        "wool": int
                    }
                },
                "completed": bool
            }, ...
        ], ...
    ],
}
```

En caso de ser un comercio con la banca, en lugar de pasar una oferta de intercambio en "trade_offer", se pasa simplemente qué material quiere entregar y a cambio de cuál.

```
{
    "trade_offer":{
        "gives": int,
        "receives": int
    },
    "harbor_trade": bool,
    "answer":{
        "cereal": int,
        "mineral": int,
        "clay": int,
        "wood": int,
        "wool": int
    }
}
```

En caso de jugar una carta de desarrollo, el objeto necesita que se le pase qué carta se ha jugado. Para ello se añade la clave "development_card_played" y a dicha clave se le asocia el objeto de la carta jugada.

```
{
    "trade_offer": "played_card",
    "harbor_trade": null,
    "development_card_played":{
         objeto variable
    }
}
```

Y cuando no se quiere comerciar, la clave "trade_offer" posee texto plano indicando que no se quiere comerciar.

```
{
    "trade_offer": "None"
}
```

Dado que es un array, pueden haber diferentes objetos a la vez, por ejemplo, un jugador comercia con otro, luego con la banca y luego decide dejar de comerciar. Ese ejemplo tendría un array con los tres posibles objetos, uno detrás del otro. El visualizador lee el valor de *trade_offer* y en función de lo que recibe, muestra un comercio entre jugadores, con la banca o simplemente pasa de fase.

build_phase: Objeto que muestra la fase de construcción. La fase de construcción es un array compuesto por diferentes objetos que cambian en función de lo que el bot haya construido durante la fase. Los posibles objetos son los siguientes:

En caso de que no hayan construcciones, la clave *building* posee un string indicando que no hay construcciones.

```
{
    "building": "None"
}
```

En caso de que se construya un pueblo, este es el objeto.

```
{
    "building": "town",
    "node_id": int,
    "finished": bool
}
```

En caso de que se construya una ciudad, el objeto es idéntico a haber construido un pueblo.

```
{
    "building": "city",
    "node_id": int,
    "finished": bool
}
```

En caso de construir una carretera, se necesita también el nodo final al que apunta la carretera, por eso se añade el apartado "road_to".

```
{
    "building": "road",
    "node_id": int,
    "road_to": int,
    "finished": bool
}
```

Cuando se construye una carta, si la construcción se completa se añaden los campos de la carta robada, para que el visualizador pueda indicar visualmente qué carta tiene qué jugador en sus manos.

```
{
    "building": "card",
    "finished": bool,
    "card_id": int,
    "card_type": int,
    "card_effect": int
}
```

En caso de que se juegue una carta, el objeto necesita que se le pase qué carta se ha jugado. Para ello se añade la clave "development_card_played" y se le pasa uno de los objetos asociados a la clave.

```
{
    "building": "played_card",
    "finished": "played_card",
    "development_card_played": {
        objeto variable
    }
}
```

development_card_played: Usado solo al inicio del turno y al final. Posee diferentes objetos en función de la carta de desarrollo jugada. El array se deja vacío en caso de que no se juegue ninguna carta de desarrollo. Los posibles objetos son los siguientes:

Al jugar una carta de puntos de victoria y tener suficientes puntos de victoria para ganar, este es el objeto que saldría:

```
{
    "played_card": "victory_point",
}
```

En caso de jugar una carta de puntos de victoria, pero carecer de los suficientes para ganar la carta se llama "failed_victory_point".

```
{
    "played_card": "failed_victory_point",
}
```

En caso de que un bot juegue una carta de caballero. La clave *robbed_player* hace referencia a la ID del jugador al que el bot ha decidido robarle un material.

```
{
    "played_card": "knight",
    "total_knights": int,
    "past_thief_terrain": int,
    "thief_terrain": int,
    "robbed_player": int,
    "stolen_material_id": int
}
```

En caso de que un bot juegue una carta de construcción de carreteras:

```
{
    "played_card": "road_building",
    "valid_road_1": bool,
    "valid_road_2": bool,
    "roads":{
        "node_id": int,
        "road_to": int,
        "node_id_2": int,
        "road_to_2": int
    }
}
```

En caso de jugar una carta de monopolio, el objeto tiene el siguiente aspecto:

```
{
    "played_card": "monopoly",
    "material_chosen": int,
    "material_sum": int,
    "hand_P0": {
        "cereal": int,
        "mineral": int,
        "clay": int,
        "wood": int,
        "wool": int
    },
    "hand_P1": {...},
```

```
    "hand_P2": {...},
    "hand_P3": {...},
}
```

En caso de que se juegue una carta de año de la cosecha, el objeto tiene el siguiente aspecto. En el apartado "hand_P0", el objeto sustituye el número "0" por el número del jugador que haya jugado la carta, de tal manera que si quien la ha jugado es el jugador 4, en el objeto quedaría "hand_P3".

```
{
    "played_card": "year_of_plenty",
    "materials_selected": {
        "material": int,
        "material_2": int
    },
    "hand_P0": {
        "cereal": int,
        "mineral": int,
        "clay": int,
        "wood": int,
        "wool": int
    },
}
```