LÓGICA PARA CIENCIAS DE LA COMPUTACIÓN

Proyecto N°1 - GO

Primer Cuatrimestre 2019

Fernandez Felipe 114784
Iarlori Federico 114986

Introducción.

GO es un juego de mesa para dos jugadores que consiste de un tablero de 19 líneas horizontales (filas) por 19 líneas verticales (columnas) y piezas llamadas piedras (fichas) de color blanco y negro. El tablero se encuentra vacío al comienzo del juego, y los jugadores, llamados blanco y negro, deben colocar piedras del color correspondiente, por turnos, en las intersecciones de las líneas verticales y horizontales. El jugador negro realiza la primera movida, luego de lo cual blanco y negro alternan. Una movida consiste en ubicar una piedra del color propio en una intersección vacía del tablero. Una vez que una piedra es ubicada ya no puede moverse, pero puede ser capturada como resultado de una jugada del jugador contrario. Al colocar una piedra, todas aquellas piedras del oponente que queden rodeadas (o encerradas) por piedras propias serían capturadas, esto es, removidas del tablero.

Captura de Piedras.

Dada una configuración del tablero, decimos que un conjunto dado A (no vacío) de piedras de color C (blanco o negro) es un conjunto de piedras rodeadas si y solo si cada piedra de A verifica que las posiciones adyacentes, vertical u horizontalmente, que la circundan contienen piedras del color contrario, o piedras de color C que también pertenecen a A (es decir, que también están rodeadas). Cuando un jugador coloca una piedra, todas aquellas piedras del oponente que quedan rodeadas (si es que alguna) serán capturadas.

Jugadas Validas.

Diremos que colocar una piedra de color C en una intersección dada constituiría suicidio si como resultado de colocarla en dicha posición se generaría un conjunto de piedras rodeadas de color C (por supuesto, entre las cuales estaría la piedra colocada), salvo que piedras contrarias también queden atrapadas, en cuyo caso no estaríamos en presencia de suicidio sino que capturaríamos las contrarios. Un jugador podría colocar una piedra en cualquier intersección vacía, con la restricción de que al colocarla en dicho lugar no cometa suicidio. Además de esto, un jugador puede pasar (saltear) su turno en cualquier momento.

Finalización del juego y puntajes.

El juego termina cuando los dos jugadores pasaron consecutivamente. El puntaje de cada jugador es el número de intersecciones que quedaron ocupadas por piedras propias, además de aquellas intersecciones vacías que quedaron encerradas o rodeadas por piedras propias (territorio). El jugador con mayor puntaje es el ganador. Observación: la definición de conjunto de intersecciones vacías rodeadas por piedras de un jugador es análoga a la que se usa para determinar la captura de piedras contrarias, especificada anteriormente.

Metodos implementados:

Puntaje Color:

Método cascara que se llama desde Javascript para pedir el puntaje de un color ingresado por parámetro. Si el tablero no está vacío buscamos el puntaje llamando al método recorrer y si esta vacío, devuelve 0.

```
puntajeColor(Board, Color, Puntaje):-
    not(vacio(Board, [0,0])),
    recorrer(Board , [0,0] , Color , []  , Puntaje).

puntajeColor(Board, _, 0):-
    vacio(Board, [0,0]).
```

Vacio:

Chequea si el tablero esta vacio, esto es no hay ninguna ficha de algun color, solo hay guiones. Si recorrimos todo el tablero y no encontramos ninguno devuelve verdadero y si lo estamos recorriendo y encontramos un guion, pedimos el siguiente y vemos si este también es guion. Sino devuelve falso.

```
vacio(_, [R,C]):-
    R= 19,
    C=19.

vacio(Board, [R,C]):-
    ficha(Board, R, C ,Ficha),
    Ficha = "-",
    siguiente(R ,C, R1, C1),
    vacio(Board, [R1, C1]).
```

Recorrer:

Recorrido del tablero, sumando 1 al puntaje si la ficha encontrada es de tu mismo color, y si se encuentra un “-” se chequea si esta encerrado por el color del parametro y se suman al puntaje la cantidad de guiones que estan encerrados.

- Caso base: terminamos de recorrer todo el tablero y el puntaje es 0.

```
recorrer(_, [R,C] , _ , _ , 0):-
    R = 19,
    C = 19.
```

- Visitamos una ficha que es de distinto color al que deseamos contar el puntaje, entonces seguimos recorriendo el tablero con la siguiente intersección del tablero.

```
recorrer(Board, [R,C], Color, Visitados , Puntaje):-
    R \= 19,
    C \= 19,
    ficha(Board, R, C, Ficha),
    sonOpuestos(Color,Ficha),
    siguiente(R, C, R1, C1),
    recorrer(Board, [R1, C1], Color, Visitados , Puntaje).
```

- Visitamos una ficha que es del mismo color al que deseamos contar el puntaje, entonces sumamos uno al puntaje y seguimos recorriendo el tablero con la siguiente intersección del tablero.

```
recorrer(Board, [R,C], Color, Visitados , Puntaje):-
    R \= 19,
    C \= 19,
    ficha(Board, R, C, Ficha),
    Color = Ficha,
    siguiente(R, C, R1, C1),
    recorrer(Board, [R1, C1], Color, Visitados , NPuntaje),
    Puntaje is NPuntaje + 1.
```

- Visitamos un guion que ya se había visitado entonces seguimos recorriendo el tablero con la siguiente intersección del tablero.

```
recorrer(Board, [R,C], Color, Visitados , Puntaje):-
    R \= 19,
    C \= 19,
    ficha(Board, R, C, Ficha),
    Ficha = "-",
    member([R,C,Ficha], Visitados),
    siguiente(R, C, R1, C1),
    recorrer(Board, [R1,C1], Color , Visitados , Puntaje).
```

- Visitamos un guion que no había sido visitado, le pedimos sus adyacentes, y con el método territorioCapturado, recorremos el tablero buscando si el conjunto de guiones al cual pertenece este guion (es decir, los recursivamente adyacentes) están encerrados y devuelve una lista con los nuevos guiones visitados y el puntaje de ese territorio de guiones capturados, luego concatenamos la lista de los nuevos visitados (NuevosVisitados) con los visitados que ya teníamos anteriormente (Visitados) en una nueva lista VisitadosTotales, y volvemos a recorrer el tablero con la siguiente intersección del tablero con los nuevos visitados. Por último el puntaje a devolver va a ser la suma entre el puntaje que nos devolvió el método territorioCapturado y la salida de la recursión de recorrer.

```
recorrer(Board, [R,C], Color, Visitados , Puntaje):-
    R \= 19,
    C \= 19,
    ficha(Board, R, C, Ficha),
    Ficha = "-",
    not(member([R,C,Ficha], Visitados)),
    adyacentes(Board,[R,C],Adyacentes),
    territorioCapturado(Board , Color , [R,C,Ficha] , Adyacentes ,  Visitados  , [] , NuevosVisitados , SubPuntajeTerritorio),
    append( Visitados  , NuevosVisitados , VisitadosTotales),
    siguiente(R, C, R1, C1),
    recorrer(Board, [R1,C1] , Color , VisitadosTotales , NPuntaje),
    Puntaje is SubPuntajeTerritorio + NPuntaje.
```

- Visitamos un guion que no había sido visitado, le pedimos sus adyacentes y al fallar el método territorioCapturado, esto quiere decir que el guion no está siendo capturado por fichas del color que estamos contando el puntaje, entonces seguimos recorriendo el tablero con la siguiente intersección y puntaje a devolver es el que sale de la recursión.

```
recorrer(Board, [R,C], Color, Visitados , Puntaje):-
    R \= 19,
    C \= 19,
    ficha(Board, R, C, Ficha),
    Ficha = "-",
    not(member([R,C,Ficha], Visitados)),
    adyacentes(Board,[R,C],Adyacentes),
    not(territorioCapturado(Board , Color , [R,C,Ficha] , Adyacentes ,  Visitados  , [] , _ , _)),
    siguiente(R , C , R1 , C1 ),
    recorrer(Board, [R1,C1] , Color , [ [R,C,Ficha] | Visitados ] , Puntaje).
```

Territorio Capturado:

- Caso base: donde ya recorrimos toda la lista de adyacentes de un guion de tal forma que la lista quedo vacía, entonces se verifica que el guion esta rodeado, se agrega a la lista de los que se están visitando actualmente y se devuelve 1 en el puntaje.

```
territorioCapturado( _ , _ , [R,C,"-"] , [] , _ , _ , [ [R,C,"-"] | [] ] ,  1).
```

- Si una de las fichas adyacentes es del color que estamos contando el puntaje, seguimos verificando por el resto de las adyacentes.

```
territorioCapturado(Board , Color , [R,C,"-"] , [ [_,_,FichaA] | Adyacentes ] , VisitadosPrevios , VisitadosActuales , NuevosVisitados  , SubPuntaje):-
    Color = FichaA,
    territorioCapturado(Board , Color , [R,C,"-"] , Adyacentes ,    VisitadosPrevios , VisitadosActuales , NuevosVisitados , SubPuntaje).
```

- Si una de las fichas adyacentes es un guion, no pertenece a la lista de los visitados por el predicado “recorrer” (es decir, los visitadosPrevios) y pertenece a los que se están visitando actualmente, entonces seguimos preguntando por el resto de las fichas adyacentes.

```
territorioCapturado(Board , Color , [R,C,"-"] , [ [Ra,Ca,FichaA] | Adyacentes ] , VisitadosPrevios , VisitadosActuales , NuevosVisitados , SubPuntaje ):-
    FichaA = "-",
    not(member( [Ra,Ca,FichaA] , VisitadosPrevios ) ),
    member( [Ra,Ca,FichaA] , VisitadosActuales ),
    territorioCapturado(Board , Color , [R,C,"-"] , Adyacentes , VisitadosPrevios , VisitadosActuales , NuevosVisitados , SubPuntaje).
```

- Si una de las fichas adyacentes es un guion, no pertenece a la lista de los visitados por el predicado “recorrer” (es decir, los VisitadosPrevios) y no pertenece a los que se están visitando actualmente, entonces pedimos los adyacentes de esta ficha y preguntamos si ella está encerrada (habiendose agregado la ficha original a la lista de VisitadosActuales para que esta ficha adyacente no pregunte luego por la ficha original) , y si es así, entonces devuelve que fichas quedaron encerradas con ella y el puntaje obtenido. Luego concatenamos estas fichas nuevas encerradas (VisitadosEncerrados) con las que ya estaban visitadas de antes (VisitadasActuales), obteniéndose así el nuevo conjunto de fichas ya visitadas, para finalmente al visitar las fichas adyacentes de la ficha original que quedaron por visitar, no estemos visitando fichas repetidas. Luego se concatenan los guiones que habían sido encerrados antes

(VisitadosEncerrados) con las nuevas encerradas (VisitadosEncerrados1), obteniéndose el conjunto final de las fichas captuadas hasta el momento, y si ambos subpuntajes son mayores a 0, se los suma y el subpuntaje final será la suma de estos 2. Si uno de los subpuntajes es 0, quiere decir que uno de los grupos de fichas no fue capturado, por ende, ninguna de las fichas que fueron llamadas recursivamente esta capturado, así entonces el predicado debe fallar.

```
territorioCapturado(Board , Color , [R,C,"-"] , [ [Ra,Ca,FichaA] | Adyacentes ] , VisitadosPrevios , VisitadosActuales , NuevosVisitados , SubPuntaje):-
    FichaA = "-",
    not(member( [Ra,Ca,FichaA] , VisitadosPrevios ) ),
    not(member( [Ra,Ca,FichaA] , VisitadosActuales ) ),
    adyacentes(Board,[Ra,Ca],AdyacentesA),
    territorioCapturado(Board , Color , [Ra,Ca,"-"] , AdyacentesA , VisitadosPrevios , [ [R,C,"-"] | VisitadosActuales ] , VisitadosEncerrados , SubPuntaje1),
    append(VisitadosActuales , VisitadosEncerrados , VisitadosActuales1),
    SubPuntaje1 > 0,
    territorioCapturado(Board , Color , [R,C,"-"] , Adyacentes , VisitadosPrevios , VisitadosActuales1 , VisitadosEncerrados1 , SubPuntaje2),
    append(VisitadosEncerrados,VisitadosEncerrados1,NuevosVisitados),
    SubPuntaje2 > 0,
    SubPuntaje is SubPuntaje1 + SubPuntaje2.
```

- Si una de las fichas adyacentes ya era miembro de la lista de visitados de “recorrer” (es decir, VisitadosPrevios), entonces la ficha original no está rodeada, terminando así con la recursión y devolviendo 0.

```
territorioCapturado( _ , _ , [R,C,"-"] , [ [Ra,Ca,FichaA] | _ ] , VisitadosPrevios , _ , [ [R,C,"-"] | [] ] , 0):-
    member( [Ra,Ca,FichaA] , VisitadosPrevios ).
```

- Si una de las fichas adyacentes es de un color opuesto al pasado por parámetro (es decir, de quien queremos contar el puntaje), entonces la ficha original no está rodeada, terminando así con la recursión y devolviendo 0.

```
territorioCapturado( _ , Color , [R,C,"-"] , [ [_,_,FichaA] | _ ] , _ , _ , [ [R,C,"-"] | [] ] , 0):-
    sonOpuestos( Color , FichaA ).
```

Siguiente:

Dada una intersección en el tablero, devuelve la próxima valida, sino devuelve fila=19 y col=19. Recorremos todas las columnas de una fila antes de recorrer la próxima fila.

- La columna pasada no es la última, por lo tanto visitamos la próxima columna de la misma fila.

```
siguiente(R, C, R1, C1):-
    R < 19,
    C < 18,
    R1 is R,
    C1 is C+1.
```

- La columna pasada es la última de la fila y la fila no es la última del tablero, por lo tanto, visitamos la primer columna de la próxima fila.

```
siguiente(R, C, R1, C1):-
    R < 18,
    C = 18,
    R1 is R+1,
    C1 is 0.
```

- Nos encontramos en la última intersección del tablero, por lo tanto no hay más que recorrer y devolvemos Fila=19, Columna=19 para diferenciar el final del tablero.

```
siguiente(R, C, R1, C1):-
    R = 18,
    C = 18,
    R1 is 19,
    C1 is 19.
```

Eliminar Capturados Adyacentes:

Chequea si al ingresar una ficha al tablero capturo a fichas de distinto color y llama al método eliminarCapturados y las "elimina" del tablero.

- Caso base: Ya recorrimos toda la lista de adyacentes, entonces el tablero a devolver es el que modificamos.

```
eliminarCapturadosEnAdyacentes(Board , _ , [] , Board).
```

- En el caso que la ficha visitada sea del color opuesto al de la ficha que ingresamos, chequeamos si esta está encerrada (Con el método suicidio que nos devuelve una lista con las fichas adyacentes también capturadas), eliminamos del tablero las fichas que nos devuelve el método suicidio, y por último seguimos recorriendo la lista de adyacentes.

```
eliminarCapturadosEnAdyacentes(Board , Color , [ [R,C,ColorA] | Adyacentes ] , Board2):-
    sonOpuestos(Color,ColorA),
    suicidio(Board , [R,C,ColorA] , Capturados),
    eliminarCapturados(Board , Capturados , Board1),
    eliminarCapturadosEnAdyacentes(Board1 , Color , Adyacentes , Board2).
```

- En el caso que la ficha visitada sea del mismo color al de la ficha que ingresamos, la descartamos y seguimos visitando la lista de adyacentes.

```
eliminarCapturadosEnAdyacentes(Board , Color , [ [_,_,ColorA] | Adyacentes ] , Board1):-
    Color = ColorA,
    eliminarCapturadosEnAdyacentes(Board , Color , Adyacentes , Board1).
```

- En el caso que la ficha visitada de adyacentes no esté encerrada, la descartamos y seguimos visitando la lista de adyacentes.

```
eliminarCapturadosEnAdyacentes(Board , Color , [ [R,C,ColorA] | Adyacentes ] , Board1):-
    not(suicidio(Board , [R,C,ColorA] , _ )),
    eliminarCapturadosEnAdyacentes(Board , Color , Adyacentes , Board1).
```

Eliminar Capturados:

Dado una lista de fichas capturadas, las "elimina" del tablero, las reemplaza con un guion. Cuando la lista está vacía entonces el tablero a devolver es el que se estaba modificando, sino, se toma una ficha de adyacentes y se reemplaza su intersección por un guion y se sigue recorriendo la lista de Capturados.

```
eliminarCapturados(Board, [] ,Board).

eliminarCapturados(Board,[ [R,C,Color] | Capturados ],Board2):-
    replace(Row, R, NRow, Board, Board1),
    replace(Color, C, "-", Row, NRow),
    eliminarCapturados(Board1 , Capturados , Board2).
```

Suicidio:

Método cascara para calcular si la ficha pasada por parámetro se encuentra encerrada y devuelve una lista, Aeliminar, con las fichas que también se encuentran encerradas.
Llama a locked con el tablero, la lista de adyacentes de la ficha a chequear, una lista vacía para guardar las fichas visitadas a lo largo del recorrido y por último la lista Aeliminar donde se devolverán las fichas encerradas.

```
suicidio(Board,  [R,C,Color] , Aeliminar):-
    adyacentes(Board , [R,C] , Adyacentes),
    locked(Board , Adyacentes , [R,C,Color] , [] , Aeliminar).
```

Locked:

Chequea si la ficha pasada por parámetro y sus adyacentes no se encuentran encerrados por fichas de distinto color. Y en el caso que estén encerrados, los agrega a la lista Aeliminar, siendo esta el parámetro de salida.

Al descartar los adyacentes que son de distinto color, si llegamos al caso donde la lista de adyacentes se vacía, nos indicaría que la ficha está rodeada por fichas de color contrario.

- Caso base: cuando la lista de adyacentes esta vacía, entonces la ficha se encuentra encerrada y se agrega a la lista de Aeliminar.

```
locked( _ , [] , [R,C,Color] , _ , [ [R,C,Color] | [] ] ).
```

- Si el color de la ficha visitada de adyacentes es opuesto al de la ficha inicial, se descarta y se sigue visitando a los demás adyacentes

```
locked(Board , [ [_,_,ColorA] | Adyacentes ] , [R,C,Color] , Visitados , Aeliminar):-
    sonOpuestos(ColorA,Color),
    locked(Board, Adyacentes , [R,C,Color] , Visitados , Aeliminar).
```

- Si el color de la ficha visitada de adyacentes es igual al de la ficha inicial, y no se había visitado anteriormente, entonces se calculan sus fichas adyacentes, se chequea si estas están encerradas (agregando a la llamada la ficha original a la lista de visitados, para que la ficha adyacente no vuelva a preguntar por ella) y si lo están se agregan las fichas a Visitados (Donde guardamos las fichas del mismo color que estamos chequeando si se encuentran encerradas o no). Se concatenan las listas de Visitados, con las fichas encerradas que se encontraron y se termina de chequear si los adyacentes a la primera ficha están encerrados con la lista concatenada como

parámetro de entrada (Lista de fichas visitadas del mismo color). Y por último se devuelve en la lista Aeliminar, la concatenación de las fichas encerradas de las dos pasadas de locked.

```
locked(Board , [ [Ra,Ca,ColorA] | Adyacentes] , [R,C,Color] , Visitados , Aeliminar):-
    Color = ColorA,
    not(member([Ra,Ca,ColorA] , Visitados)),
    adyacentes(Board , [Ra,Ca] , AdyacentesA),
    locked(Board , AdyacentesA , [Ra,Ca,ColorA] , [ [R,C,Color] | Visitados ] , Aeliminar1),
    append(Visitados,Aeliminar1,VisitadosYeliminados),
    locked(Board ,  Adyacentes , [R,C,Color] , VisitadosYeliminados , Aeliminar2),
    append(Aeliminar1,Aeliminar2,Aeliminar).
```

- Si son del mismo color y ya había sido visitado, se descarta y se sigue recorriendo con los demás adyacentes.

```
locked(Board , [ [Ra,Ca,ColorA] | Adyacentes] , [R,C,Color] , Visitados , Aeliminar):-
    Color = ColorA,
    member([Ra,Ca,ColorA] , Visitados),
    locked(Board ,  Adyacentes , [R,C,Color] , Visitados , Aeliminar).
```

Adyacentes:

El método adyacentes devuelve en el tercer parámetro (LValida) una lista con los adyacentes validas a la ficha que se encuentra en la posición (R, C) ingresada por parámetro. Denominamos adyacentes validos a las intersecciones adyacentes que se encuentran dentro del tablero. Y en la lista guardamos listas de tres elementos [R, C, Ficha] con la correspondiente fila, columna y color de piedra o guion de la intersección adyacente.

```
adyacentes(Board,[R,C],Lvalida):-
    C1 is C-1,
    C2 is C+1,
    R1 is R+1,
    R2 is R-1,
    ficha(Board,R2,C,Fup),
    ficha(Board,R1,C,Fdown),
    ficha(Board,R,C1,Fleft),
    ficha(Board,R,C2,Fright),
    LAdyacentes = [ [R2,C,Fup] , [R1,C,Fdown] , [R,C1,Fleft] , [R,C2,Fright] ],
    adyacentesValidos(LAdyacentes,Lvalida).
```

Adyacentes Validos:

El método dada una lista de intersecciones, devuelve una lista nueva solo con las intersecciones que contengan alguna ficha de color o un guion. Descarta las intersecciones que contengan como color ".", estas son las que se encuentran fuera del tablero.

```
adyacentesValidos([],[]).

adyacentesValidos([ [R,C,Color] | Lad ] , [ [R,C,Color] | LNueva ]):-
    Color \= ".",
    adyacentesValidos(Lad,LNueva).

adyacentesValidos([ [_,_,Color] | Lad ] , LNueva):-
    Color = ".",
    adyacentesValidos(Lad,LNueva).
```

Ficha:

Dado un tablero y una fila y columna pasadas por parámetro, devuelve en el parámetro Ficha, el elemento que se encuentra en esa intersección, Piedra Blanca "w", Piedra Negra "b" o guion "-". Y en el caso que la intersección se encuentre fuera del tablero, devuelve un Punto "."

```
ficha(Board,R,C,Ficha):-
    dentroDelTablero(R,C),
    nth0(R,Board,Fila),
    nth0(C,Fila,Ficha).

ficha(_,R,C,Ficha):-
    not(dentroDelTablero(R,C)),
    Ficha = ".".
```

Dentro Del Tablero:

Chequea si la intersección entren la fila y columna (R, C) pasadas por parámetro se encuentra dentro del tablero.

```
dentroDelTablero(R,C):-
    R>=0,
    C>=0,
    R<19,
    C<19 .
```

Son Opuestos:

Chequea si las dos fichas ingresadas por parámetro son colores opuestos o no. Diferenciando entre Blanco y Negro.

```
sonOpuestos(Color1,Color2):-
    Color1 = "w",
    Color2 = "b".

sonOpuestos(Color1,Color2):-
    Color1 = "b",
    Color2 = "w".
```

Interfaz web:

Las modificaciones que realizamos en el documento go.js, en el lenguaje de programación Javascript fueron para representar el resultado final de la partida luego de que ambos jugadores hayan pasado su turno consecutivamente. Y son:

- Agregamos dos variables puntajeNegro y puntajeBlanco inicializadas con -1, para podes guardar el valor del puntaje que nos devuelve el método de Prolog: puntajeColor(Board, Color, Puntaje).
  Los inicializamos con -1 para poder chequear cuando estos valores fueron modificados.
- Agregamos una variable contadorTurno, para llevar la cuenta de los turnos que se pasaron sucesivamente.

Función passTurn():

Se invoca cuando un jugador pasa un turno, y en el turno anterior el jugador contrario también paso el turno (contadorTurno=1), entonces se llama al método de Prolog puntajeColor, el cual nos devuelve en la variable Puntaje, el puntaje del color blanco que obtenemos en la función handleSuccess(response) de Javascript. Y luego desde la misma función pedimos el puntaje del color negro, obtenemos su valor y mostramos el resultado final de la partida.
En cambio si el jugador del turno anterior no había pasado el turno, la variable contadorTurno pasa a valer 1 y continua la partida.

```
function passTurn(){

    if(contadorTurno == 1){

        const s = "puntajeColor(" + Pengine.stringify(gridData) + ","+ Pengine.stringify("w")+", Puntaje)";
        pengine.ask(s);
        contadorTurno=2;
    }
    else {
        contadorTurno = 1;
        switchTurn();
    }
}
```

Modificaciones en la función handleSuccess:

Si tuvo éxito la consulta que se realizó sobre el método puntajeColor (Board, Color, Puntaje), entonces response.data[0].Puntaje tendría que estar definido, y ser el puntaje del color que indicamos en la consulta. Por esto, chequeamos que el puntaje devuelto este definido. Luego si el puntaje del color blanco no fue modificado (puntajeBlanco === -1), esto indica que la consulta sobre su puntaje había sido realizada pero no teníamos su valor, asique asignamos el valor a la variable puntajeBlanco y luego consultamos a Prolog por el puntaje del negro.
Al haberse ejecutado con éxito la consulta sobre el puntaje negro, asignamos a la variable puntajeNegro el valor obtenido y luego comparamos ambos valores para ver quien fue el ganador o si hubo empate y mostramos una alerta por pantalla indicando el resultado y los respectivos puntajes. Luego llamamos al método Prolog: emptyBoard para reiniciar la partida y los valores de los puntajes.

En el caso que un jugador haya pasado su turno, pero el siguiente no, entonces la variable contadorTurno vuelve a su estado inicial =0.

```
function handleSuccess(response) {

    if(response.data[0].Board !== undefined){
        gridData = response.data[0].Board;
        for (let row = 0; row < gridData.length; row++)
            for (let col = 0; col < gridData[row].length; col++) {
                cellElems[row][col].className = "gridCell" +
                    (gridData[row][col] === "w" ? " stoneWhite" : gridData[row][col] === "b" ? " stoneBlack" : "") +
                    (latestStone && row === latestStone[0] && col === latestStone[1] ? " latest" : "");
            }
        switchTurn();

    }
    else{

        if(response.data[0].Puntaje !== undefined){

            if(puntajeBlanco === -1){

                puntajeBlanco = response.data[0].Puntaje;
                const s = "puntajeColor(" + Pengine.stringify(gridData) + ","+ Pengine.stringify("b")+", Puntaje)";
                pengine.ask(s);

            }else{

                puntajeNegro = response.data[0].Puntaje;
                if(puntajeNegro ==puntajeBlanco){
                    alert("La partida finalizo en empate, el puntaje de ambos fue: "+ puntajeBlanco);
                }
                else {
                    var ganador = (puntajeBlanco>puntajeNegro ? "Blanco" : "Negro");
                    alert("El ganador es "+ ganador + " \nEl puntaje del Blanco es: "+ puntajeBlanco + "\nEl puntaje del Negro es :" +  puntajeNegro);
                }
                puntajeBlanco = -1;
                puntajeNegro = -1;
                pengine.ask('emptyBoard(Board)');

            }

        }
    }

    contadorTurno = 0;
}
```