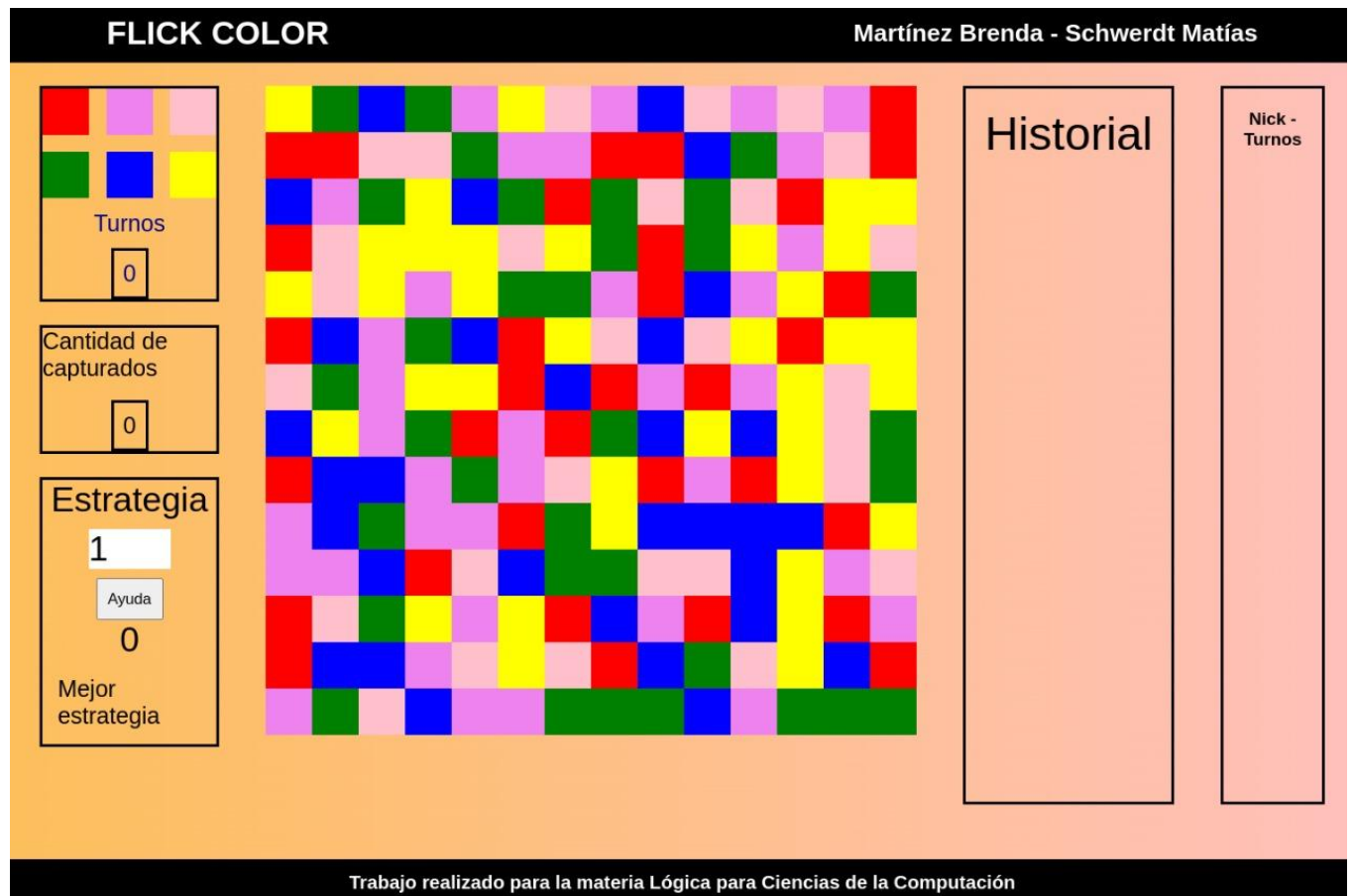


Proyecto: Flick Color

13/06/2022



Comisión 25

Integrantes:

- Brenda Belén Martínez Ocampo (LU: 111106)
- Matías David Schwerdt (LU:111667)

Ayudante a cargo: Diego



Índice

MODIFICACIONES DE LA PRIMERA ENTREGA	3
VISIÓN GENERAL	4
ARQUITECTURA DEL PROYECTO	5
FUNCIONALIDADES DEL PROYECTO	6
TRABAJO EN REACT	7
TRABAJO EN PROLOG	9
Estrategia de resolución	9
Predicados implementados	9
MANUAL DE USUARIO	11
EXTRAS	19
Raspberry Pi	19
Frontend	23
Records de turnos	24
CONCLUSIÓN	26



MODIFICACIONES DE LA PRIMERA ENTREGA

- **Trabajo en prolog**

Ahora las celdas adyacentes las obtenemos usando el código de Mauro y así tenemos la eficiencia que se necesita para desarrollar la segunda etapa del proyecto.

El predicado flick ahora tiene 8 parámetros en vez de 6, ya que ahora recibimos la lista de celdas capturadas y retornamos la nueva lista de celdas capturadas tras realizar un flick.

Creamos los predicados “iniciarConOrigenSeleccionado” e “iniciarConOrigenDefault”, para poder modelar los casos de celda inicial.

- **Errores solucionados:**

No usamos ningún assert en el predicado flick, nos manejamos siempre con listas solucionando el problema de que las celdas capturadas no permanezcan aún refrescando la página.

Removimos algunos predicados innecesarios, como por ejemplo mismoColor e insertarEnLista.

Ahora la cantidad de capturados se empieza a contar desde que se selecciona la celda origen.



VISIÓN GENERAL

El proyecto que desarrollamos es una versión del juego “Flick Color”, el cual consta de una grilla de 14 filas por 14 columnas, donde cada celda está pintada de 6 posibles colores.

Cuando el usuario presione un botón de color X, en el tablero de botones situado a la izquierda, se pintarán todas las celdas capturadas de ese color , y se agregan las celdas adyacentes que sean de ese mismo color. El objetivo del juego es pintar todo el tablero de un mismo color en la menor cantidad de turnos posibles.

El usuario puede solicitar una ayuda, indicando la cantidad de movimientos y presionando el botón ayuda, podrá visualizar la cantidad de celdas capturadas y la secuencia de colores. Esta ayuda puede pedirse en cualquier momento, todas las veces que el usuario desee.

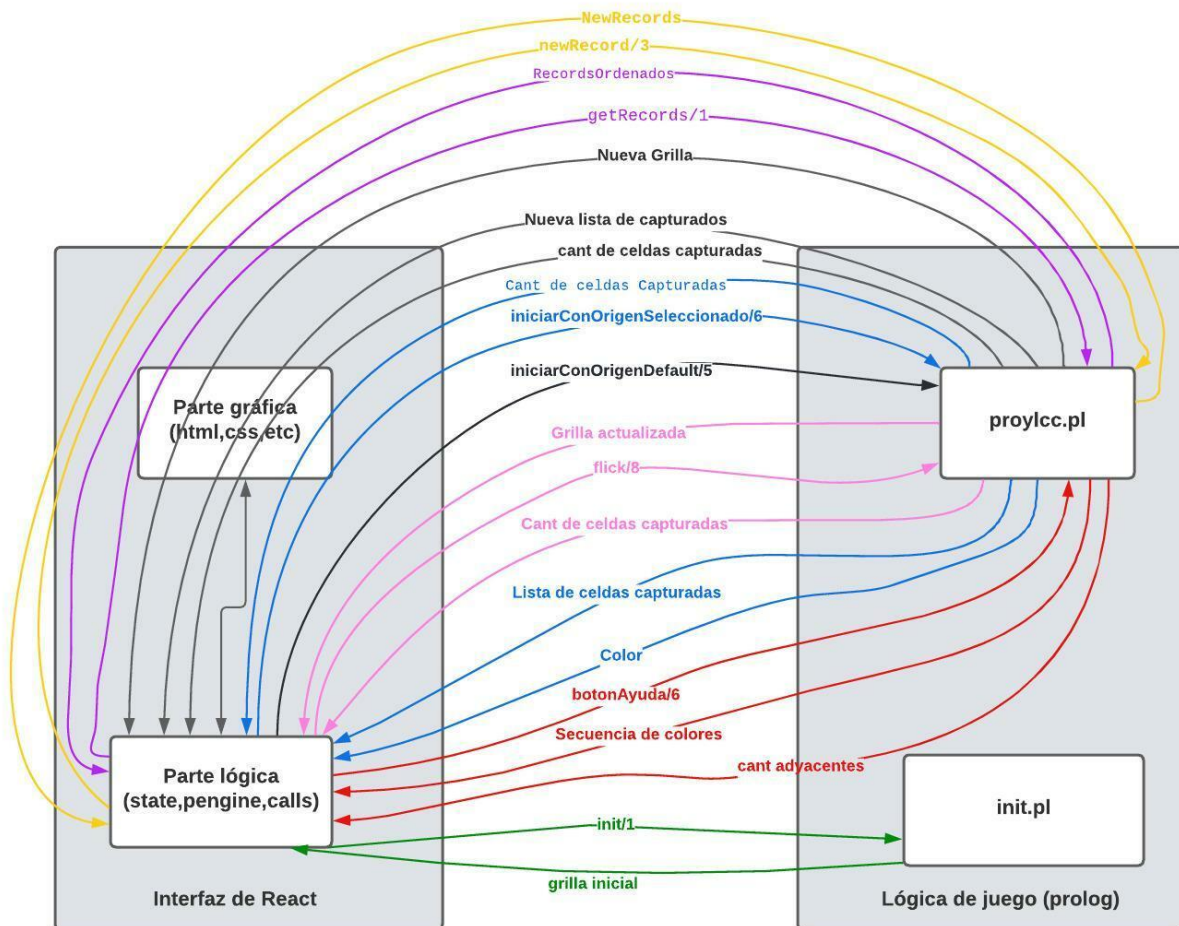
El frontend lo hemos realizado con HTML, CSS y el lenguaje de programación Javascript, utilizando el framework React.

Para el backend utilizamos Prolog, el cual es un lenguaje de programación lógico e interpretado que hemos estudiado en la materia.

Para llevarlo a cabo, partimos de una versión base, la cual fuimos modificando y programando en los archivos Game.js ,Square.js, Board.js, index.css y proylcc.pl .

ARQUITECTURA DEL PROYECTO

En el siguiente diagrama se ve reflejada la interacción entre React y Prolog.



FUNCIONALIDADES DEL PROYECTO

Al empezar a jugar se le solicita al usuario que ingrese su nick, el cual debe empezar con letra minúscula este nombre será utilizado para guardar en la tabla de récords una vez que se termine la partida.

Se puede seleccionar una celda desde donde se iniciará a jugar y esta se distinguirá con un borde negro, en caso de no seleccionar una celda, por defecto empezará el juego desde (0,0) ubicada en la esquina superior izquierda.

A la izquierda del tablero se encuentra la tabla de botones que permite elegir con qué color jugar. Debajo de dicho tablero creamos el panel de turnos realizados, es decir la cantidad de veces que se ha seleccionado un color. También creamos el panel que calcula la cantidad de celdas capturadas.

Hicimos un historial de jugadas, ubicado a la derecha del tablero, en donde se podrá observar todos los colores seleccionados durante la partida.

En el panel de la izquierda debajo de la cantidad de capturados, en la sección de estrategia, se encuentran todos los componentes gráficos para solicitar una ayuda. El usuario puede seleccionar la profundidad de la estrategia, es decir un número entre 1 y 7, para luego presionar el botón ayuda. Mostramos un cartel indicando que se está calculando la mejor estrategia y luego otro indicando que la mejor estrategia fue encontrada. Luego se verá reflejado la cantidad de capturados y la mejor estrategia de colores. En caso de que el usuario seleccione la ayuda previamente a seleccionar el origen se le mostrará un cartel indicando el error y en el caso que el usuario ingrese una profundidad fuera de rango, es decir menor 1 o mayor a 7 se le indicará el error también mediante un cartel.

A la derecha del historial se encuentra el panel de records, en donde se verán reflejadas las mejores puntuaciones, junto con el nick correspondiente que el jugador escribió al iniciar la partida.

El juego finaliza cuando el jugador logra pintar todo el tablero de un mismo color, es decir logra pintar las 196 celdas. **Creamos un cartel para comunicar esa victoria, con una animación, la cantidad de turnos en la cual logró dicha victoria y un botón “ok” que al presionarlo reinicia la partida**

TRABAJO EN REACT

Para la implementación en React simplemente pedimos que especifique los valores mantenidos en el estado del componente principal y su propósito, a alto nivel, además de las consultas realizadas a Prolog, y cómo se actualiza el estado, también a alto nivel, a partir de las respuestas obtenidas. Además, deberá escribirse una sección que explique brevemente los pasos requeridos para interactuar con la interfaz.

Sólo hemos modificado el archivo Game.js, index.css y index.html.

A continuación se muestran todas las variables que hay en el state en el archivo Game.js.

Para ésta segunda etapa del proyecto, tenemos la variable “ayudaSecuenciaColores” la cual inicialmente es un arreglo vacío, y la variable “ayudaCapturados” la cual inicialmente es 0.

Además ahora tenemos las variables “nombreJugador” y “records”, las cuales usamos para mantener una base de datos con los records en el panel de la derecha.

```
1  this.state = {
2    turns: 0,
3    grid: null,
4    complete: false, // true if game is complete, false otherwise
5    waiting: false,
6    cantidadDeCapturados: 0,
7    historial: [], // historial de colores
8    origen: undefined, // celda de origen
9    listaCapturados: [],
10   ayudaSecuenciaColores: [],
11   ayudaCapturados: 0,
12   nombreJugador: undefined,
13   records: [], // tabla de puntuaciones
14  };
```

A continuación, se puede visualizar el bloque de código en el que se realiza la consulta a Prolog para poder obtener la mejor secuencia de colores y la cantidad de celdas capturadas.

```
1  const gridS = JSON.stringify(this.state.grid).replaceAll("'", "");
2  const origen = JSON.stringify(this.state.origen).replaceAll("'", "");
3  const capturados = JSON.stringify(this.state.listaCapturados).replaceAll("'", "");
4  const queryS = "botonAyuda(" + gridS + "," + origen + "," + capturados + ","
5                  + profundidad + ", SecuenciaColores, NewCantidadAdyacentes)";
6  this.setState({
7    waiting: true
8  });
9  this.pengine.query(queryS, (success, response) => {
10    if (success) {
11      this.setState({
12        waiting: false,
13        ayudaSecuenciaColores: response['SecuenciaColores'],
14        ayudaCapturados: response['NewCantidadAdyacentes'],
15      });
16      Swal.close()
17      Swal.fire({
18        position: 'center',
19        icon: 'success',
20        title: '¡Mejor estrategia encontrada!',
21        showConfirmButton: false,
22        timer: 2000
23      })
24    } else {
25      this.setState({
26        waiting: false
27      });
28    }
29  });
```

En la siguiente imagen se puede visualizar el código que muestra en la interfaz todos los componentes necesarios para obtener la mejor estrategia.

```
1  <div className="estrategia">
2    <div className="profundidadLab">Estrategia</div>
3    <input className="profundidadNum" type="number" id="profundidad" min="1" max="7" defaultValue="1"/>
4    <button className="BotonAyuda" onClick={() => this.handleHelp()}>Ayuda</button>
5    <div className="capturadosEstrategia">{this.state.ayudaCapturados}</div>
6    <div className="ayudaLab">Mejor estrategia</div>
7    <div className="stateAyuda">{this.state.ayudaSecuenciaColores.map((colorS,i)=>
8      <Square
9        className={"squaresAyuda"}
10        value={colorS}
11        key={i}
12      />)}
13    </div>
14  </div>
```


TRABAJO EN PROLOG

Estrategia de resolución

Para encontrar la mejor secuencia de colores, vamos generando un árbol de posibles tableros tras realizar un flick.

A lo largo del desarrollo del proyecto, en múltiples ocasiones utilizamos un parámetro compuesto por 5 variables, el cual tiene el siguiente formato:

[Grilla, Origen, ListaCapturados, SecuenciaColores, ProfundidadActual]

Tener éste bloque de información es de suma utilidad para ir generando el árbol de tableros, y en todo momento poder hacer un flick, así como también tomar decisiones con respecto a la poda del árbol, y a saber cuál bloque es mejor que otro en base a la cantidad de elementos que posee la ListaCapturados. La variable ProfundidadActual nos indica a qué nivel del árbol pertenece ese bloque.

Predicados implementados

botonAyuda/6

Primero se llama al predicado estrategia/3, el cual nos devuelve todos los resultados posibles (más abajo se detalla su funcionamiento).

Luego llamamos al predicado mejorResultado/2, el cual de todos los resultados obtenidos previamente, obtenemos el mejor de todos, es decir, el resultado que más celdas adyacentes tiene.

Finalmente, retornamos la secuencia de colores y la cantidad de celdas adyacentes del mejor resultado previamente obtenido.

estrategia/3

Es un predicado recursivo, el cual está definido 4 veces.

La primera para contemplar el caso base, en el cual ya no hay que seguir iterando.

La segunda definición contempla el caso en el que todas las celdas del tablero han sido capturadas, es decir, el jugador ha ganado la partida.

En el tercer caso entramos cuando ya recorrimos exhaustivamente toda la profundidad.

En el cuarto caso obtenemos el bloque de información del nivel, lo agregamos a la lista que estamos recorriendo, para luego poder contemplarlo en una futura iteración.



getInformacionDeNivel/2

Utilizando un findall/3 obtenemos toda la información del nivel actual del árbol de tableros. Se realiza un flick de cada color, exceptuando el color actual de las celdas capturadas. Además, se descartan los flicks de un color que no cambie la cantidad de celdas capturadas.

mejorResultado/2

Tomamos como mejor bloque de información al primero de la lista, y luego llamamos al siguiente predicado para obtener el mejor.

mejorResultadoAux/3

Se llama recursivamente hasta encontrar al mejor bloque de información, es decir, al que tenga mayor cantidad de celdas capturadas.

getMejorSecuencia/3

Dados 2 bloques de información, se retorna el que tenga mayor cantidad de celdas capturadas.

getRecords/1

Se realiza un findall guardando todos los ganadores y su puntuación en una lista. Luego la lista se ordena en base a la puntuación, usando el predicado predefinido sort/4

newRecord/3

Dado un nombre de usuario y una puntuación, registra un nuevo récord.

Si el usuario ya existe en la base de datos, solo se registra su puntuación si es mejor que la que ya existía.

Si el usuario no existe, entonces se registra por primera vez.

Retorna la lista de récords actualizada.

MANUAL DE USUARIO

Al empezar a jugar Flick-Color se mostrará este cartel para que ingrese su nombre, y así poder guardar su puntuación en el registro de mejores puntuaciones, el nombre debe empezar en letra minúscula. Una vez ingresado el nombre se debe presionar "ok" para comenzar a jugar.

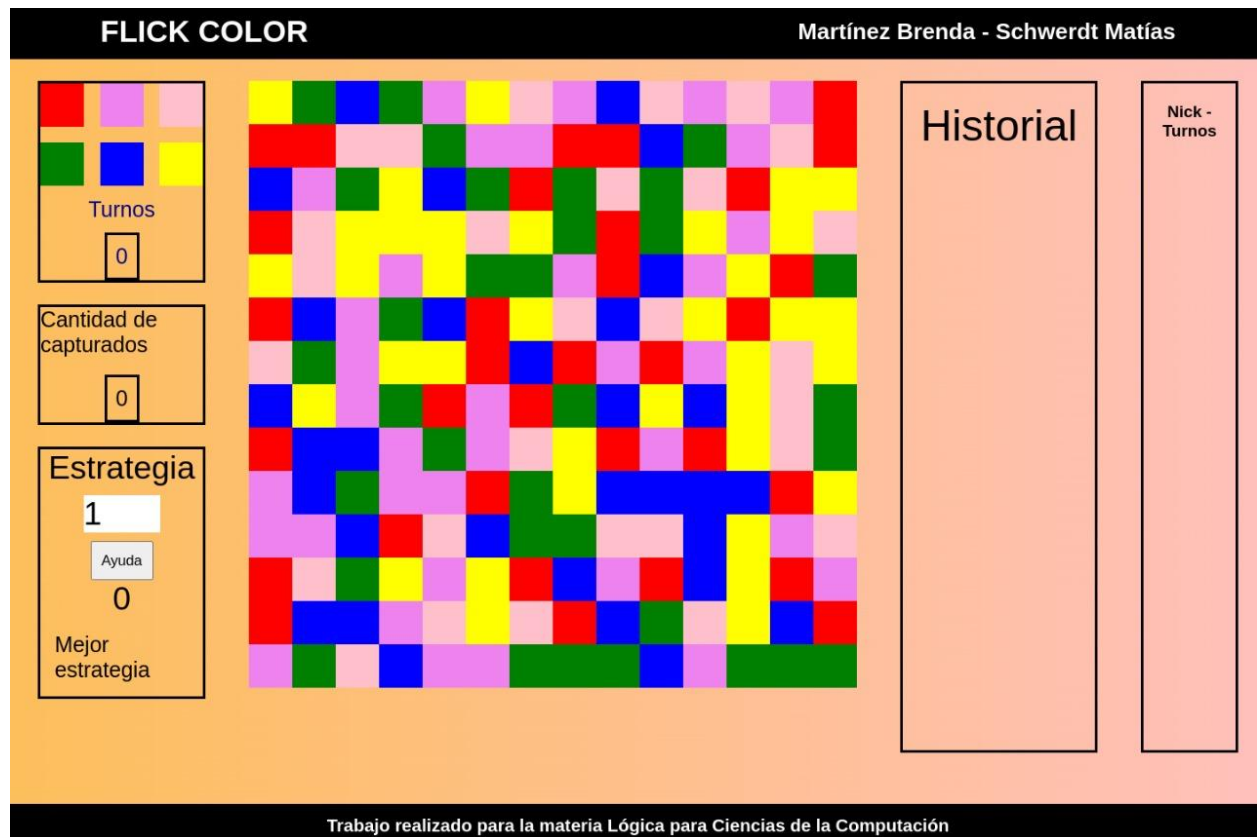


Luego de esto, podrá observar el tablero de juego en la parte central. A su izquierda se encuentran los colores que deberá ir seleccionando durante la partida y debajo de ellos se irá mostrando los turnos realizados, es decir la cantidad de veces que seleccione un color, que inicialmente se encuentra en 0.

También en la parte izquierda del tablero se puede observar la cantidad de capturados, es decir la cantidad de celdas capturadas durante toda la partida, inicialmente se encuentra en 0.

Por último en dicha sección de la izquierda del tablero se encuentra detallada la estrategia, en donde el usuario podrá solicitarla en cualquier momento, indicando la cantidad de movimientos que desea para calcular dicha estrategia, siendo este número mayor a 1 y menor a 7, debajo se encuentra el botón ayuda, el cual deberá presionar para recibir dicha ayuda, y a continuación verá detallada la cantidad de celdas capturadas y la secuencia de colores que generó la estrategia.

A la derecha del tablero el usuario podrá observar el historial de jugadas que se irán mostrando en tiempo real la selección de colores que fue usando. A su derecha se encuentra el ranking de mejores jugadas en donde se irán guardando los jugadores con su nick y su puntuación.

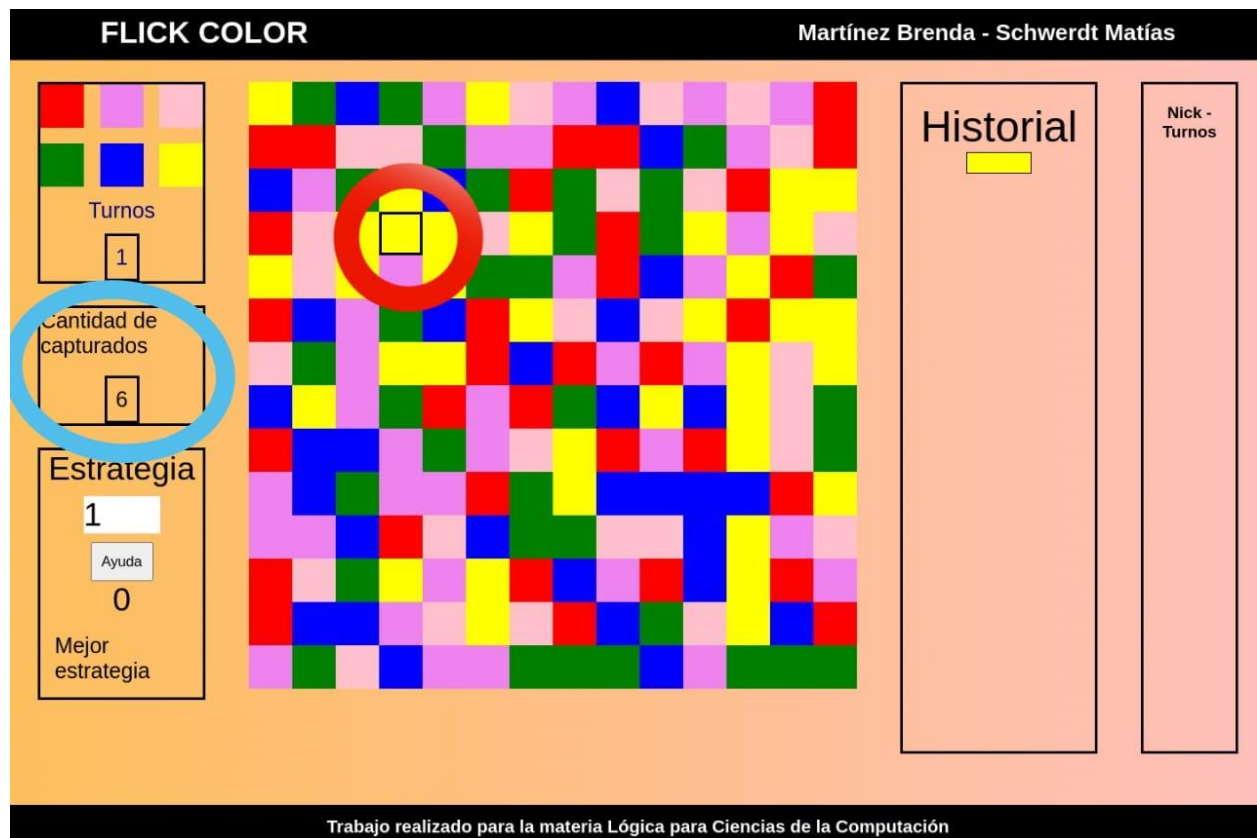


Al comenzar a jugar el usuario tiene dos opciones:

1. seleccionar una celda origen, cualquiera del tablero (ver imagen 1)
2. no seleccionar un origen y por defecto el juego empezará en la celda (0,0).(ver imagen 2)

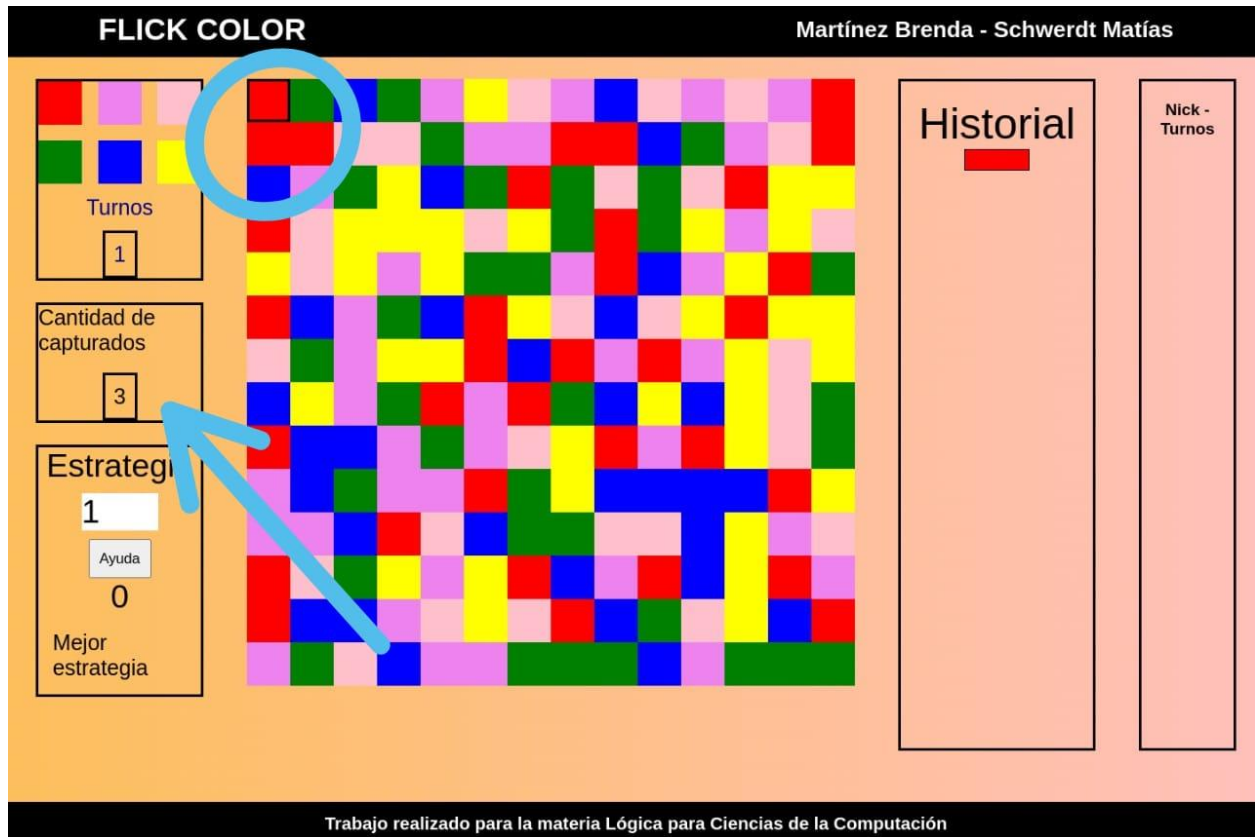
1.si selecciona un celda origen esta se marcara con borde negro, y el usuario podra visualizar la cantidad de turnos actualizada a 1. la cantidad de capturados, en este caso actualizado a 6, y el historial mostrando el color de la celda origen.

(imagen 1)

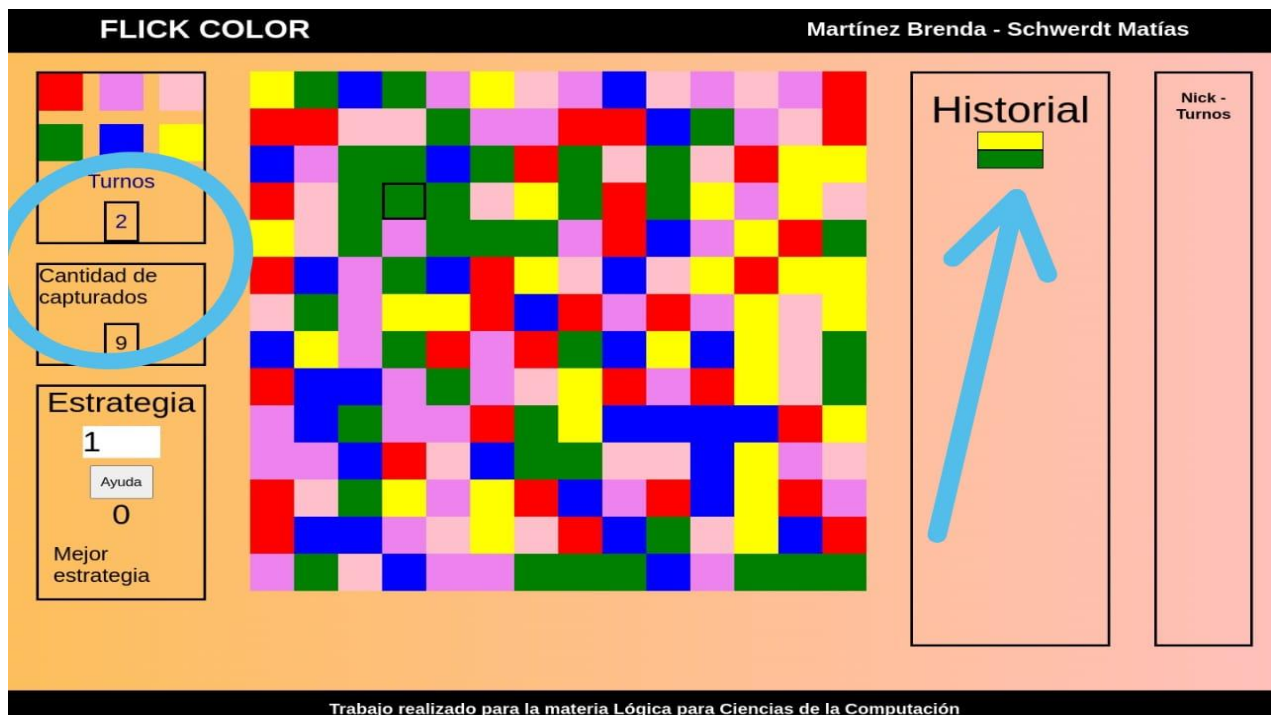


2.si el usuario no selecciona selecciona un celda origen, al elegir un color de la tabla de la izquierda se pintara de dicho color la celda 0,0 ubicada en la esquina superior izquierda, ademas de marcarse con un borde negro. También se actualizarán los turnos a 1, la cantidad de capturados, en este caso se actualizó a 3, y el historial mostrando el color seleccionado, en este caso rojo.

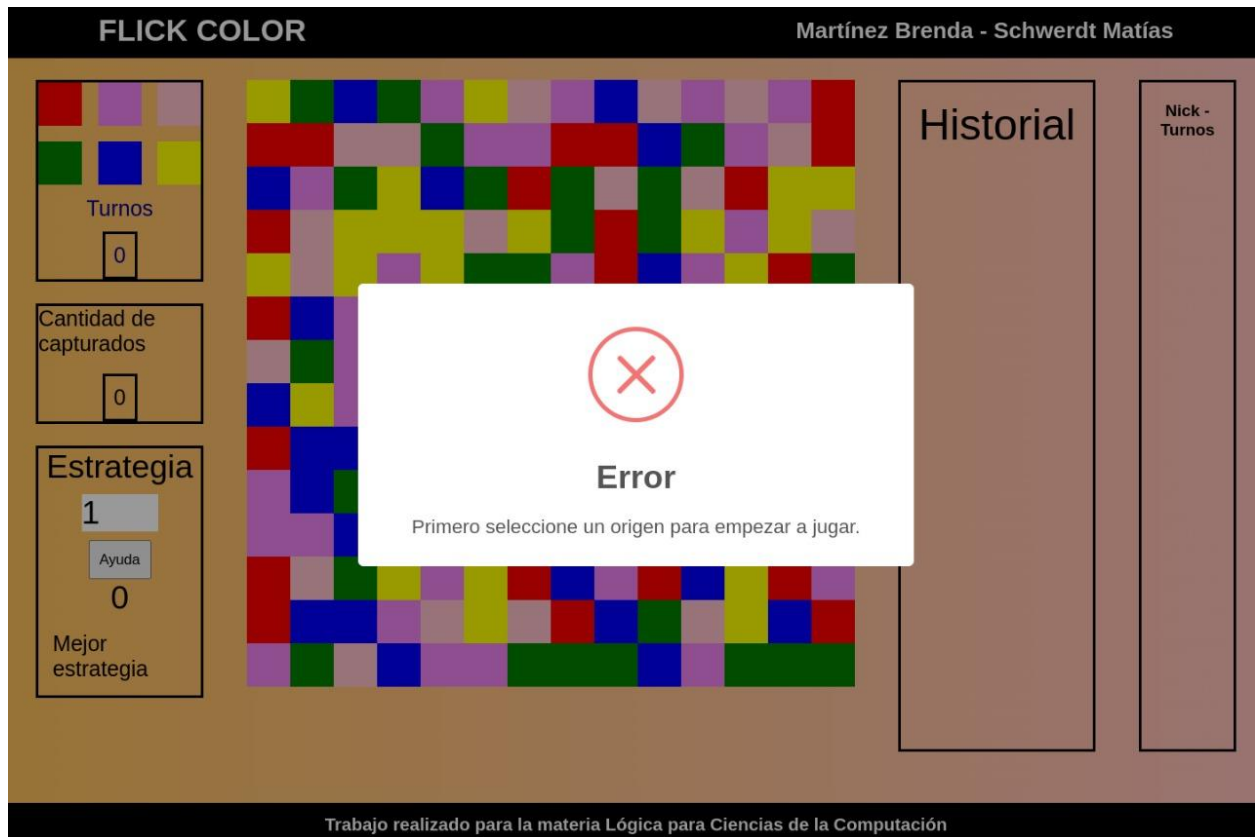
(imagen 2)



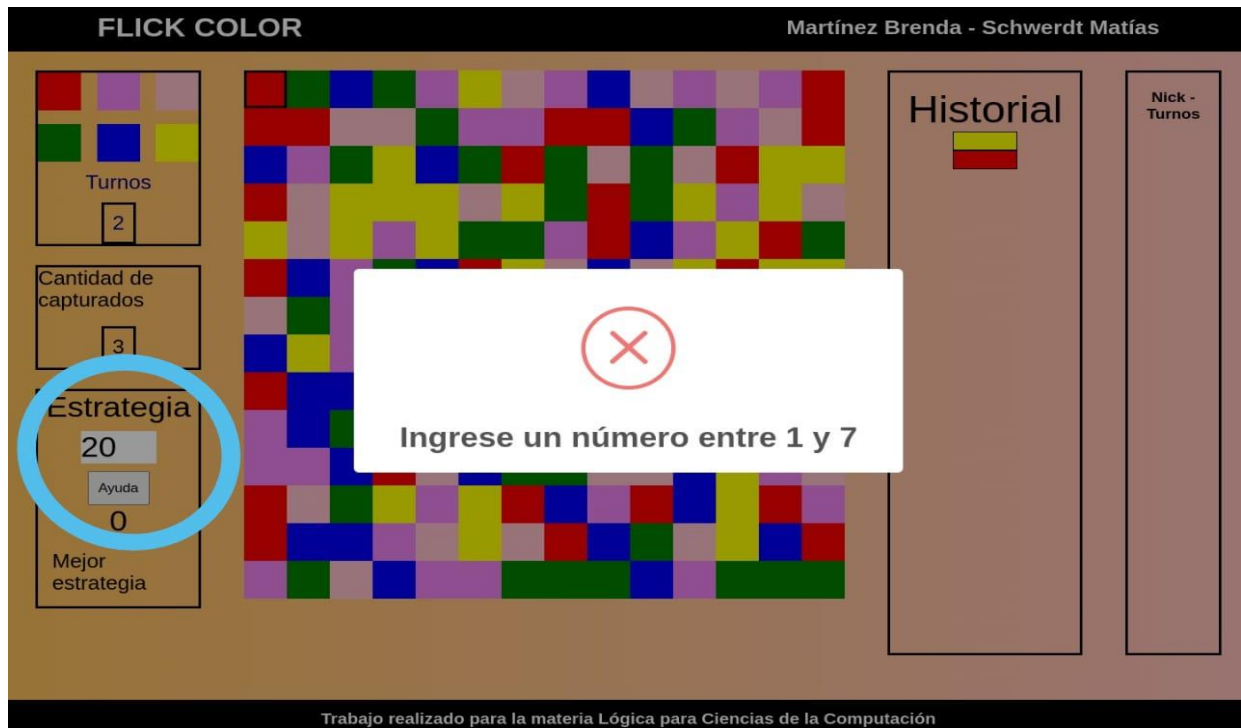
A medida que vaya jugando Flick-Color se irán actualizando tanto los turnos, cantidad de capturados como el historial como se ve en la imagen:



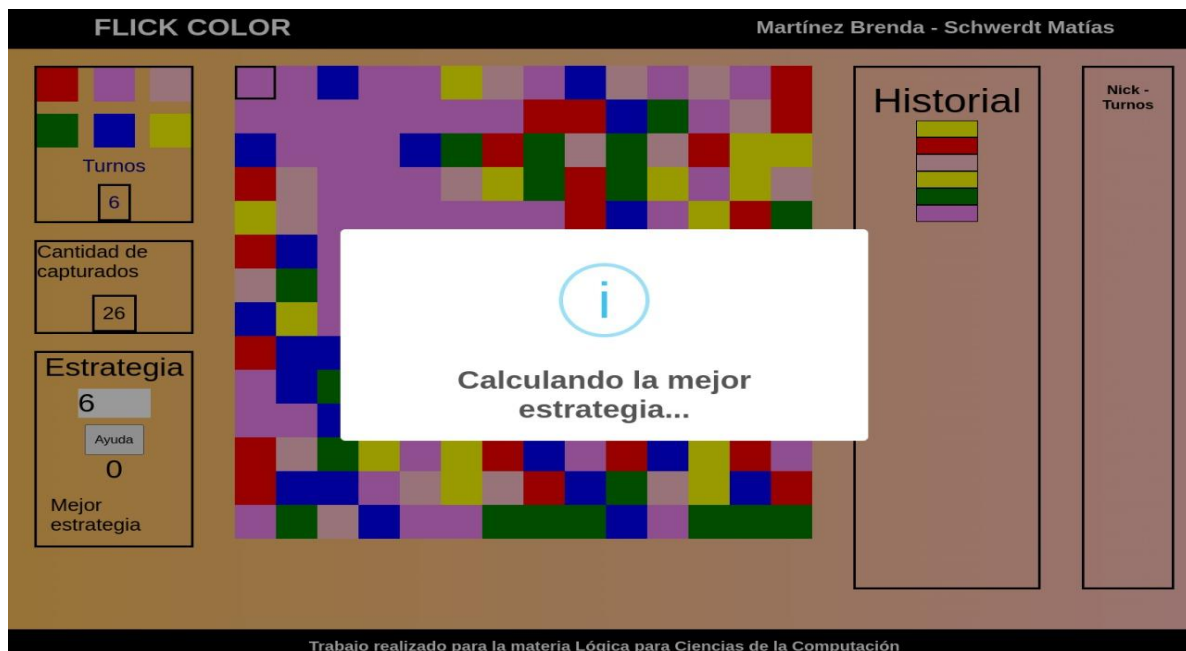
Si se solicita una ayuda previamente a seleccionar una celda origen se le mostrará un cartel indicando que esto no es posible.



Si se solicita la ayuda con un número fuera del rango, es decir un número menor a 1 ó mayor a 7, se le mostrará un cartel indicando que ingrese un número entre 1 y 7.



Al solicitar una ayuda, con un número dentro del rango, y ya habiendo seleccionado una celda origen, se le mostrará un cartel indicando que se está calculando la mejor estrategia.



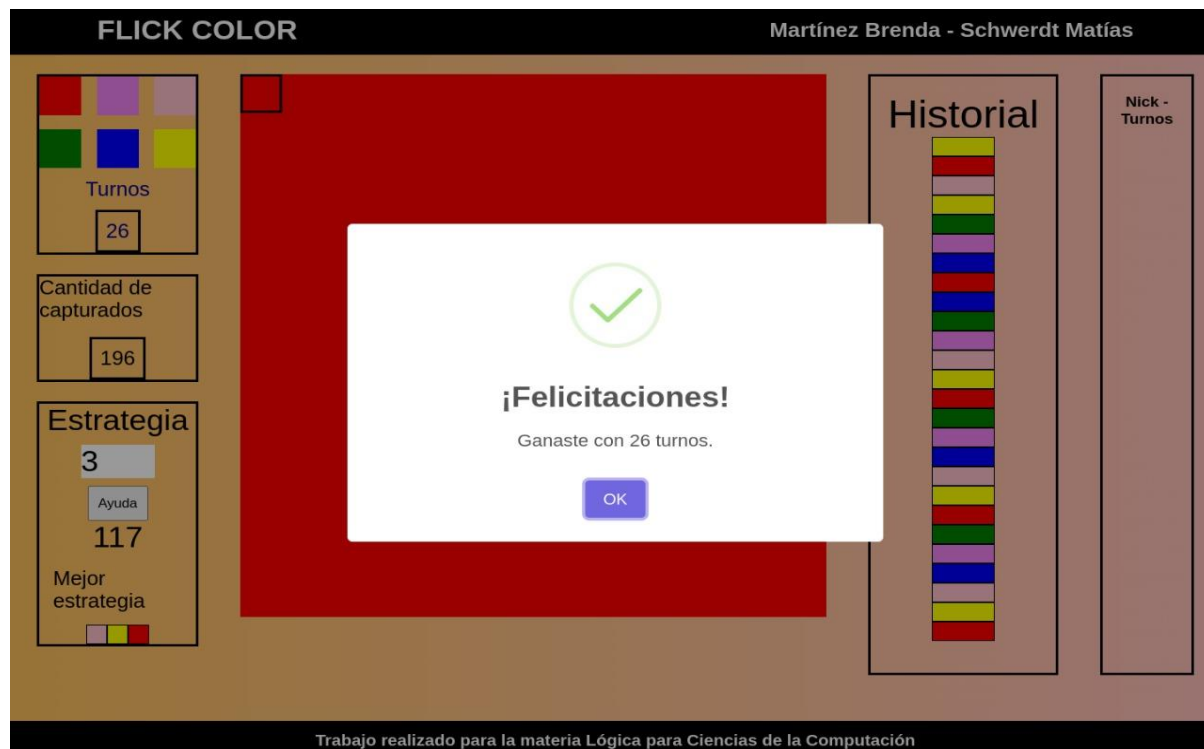
Luego de esto, se le indicará que la mejor estrategia fue encontrada mostrando la cantidad de capturados con dicha ayuda, en este caso es 99, y la secuencia de colores a seguir para lograr la mejor estrategia.



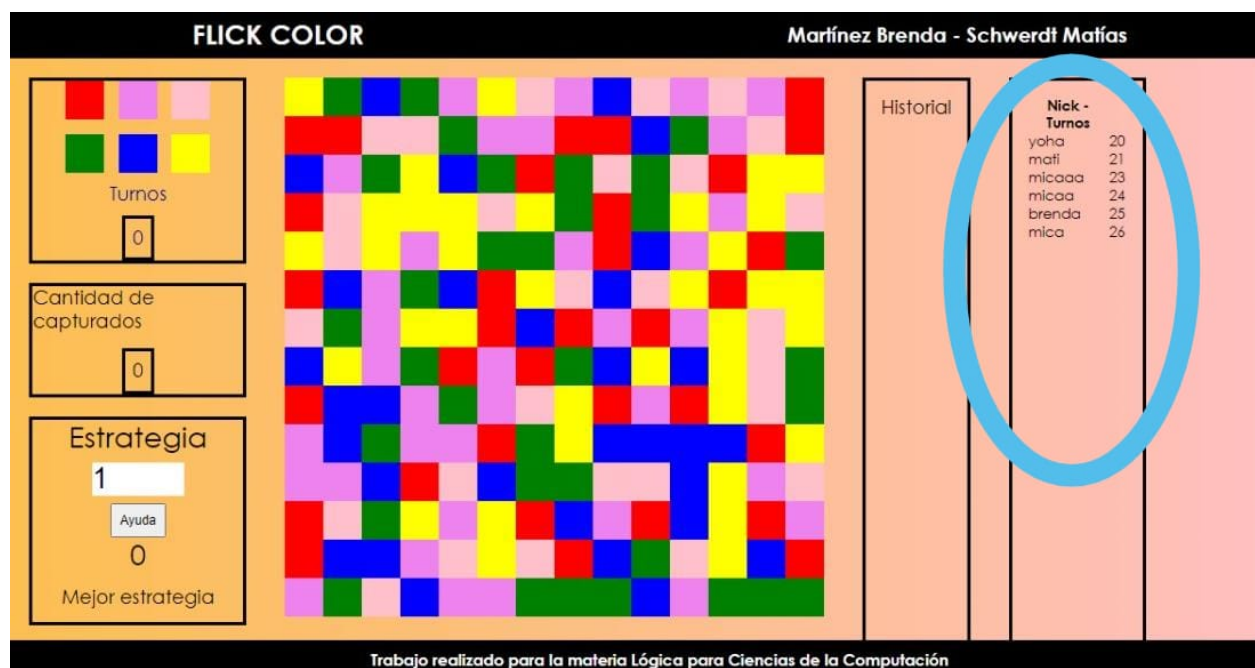
La ayuda la podrá solicitar cuantas veces desee durante la partida



El juego finaliza cuando logra pintar todo el tablero de un mismo color, si esto sucede podrá ver un cartel indicando la victoria y en cuantos turnos lo logró.



Al presionar ok, se refresca la página y podrá ver la puntuación que hizo, junto con su nick en la parte derecha de la pantalla. Allí se irán registrando los mejores puntajes. En caso de utilizar el mismo nick se registra la mejor puntuación asociada a ese nick.



EXTRAS

Raspberry Pi

Tenemos una Raspberry Pi modelo 4b. De ahora en adelante con la palabra “Raspi” haremos referencia a “Raspberry Pi”.

Para más información al respecto les dejamos un par de links de interés:

- [Página oficial de la Raspberry Pi](#)
- [Artículo de Wikipedia](#)

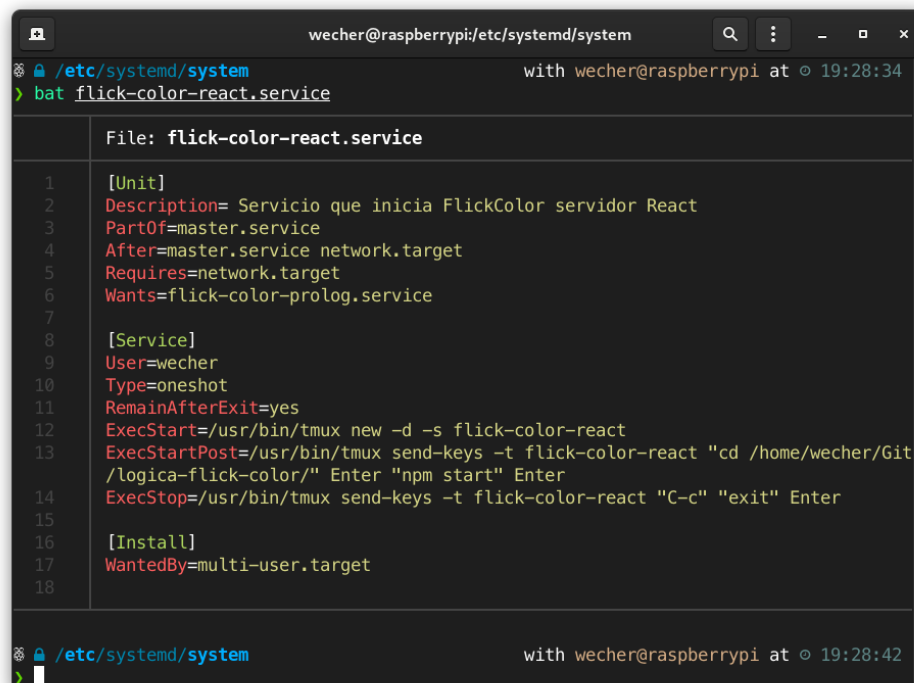
En el router tenemos el puerto abierto para poder acceder a la Raspi. Además, vinculamos la IP pública a un dns. De ésta forma, se puede acceder al juego ingresando a la siguiente página web:

<http://flickcolor.ddns.net>

Por lo tanto, el juego ahora se puede jugar desde cualquier PC, celular o tablet. Basta solamente con tener una conexión a internet para entrar al juego.

En la Raspi, para administrar la ejecución del juego utilizamos [Systemd](#).

A continuación mostramos el servicio que levanta el servidor de React.



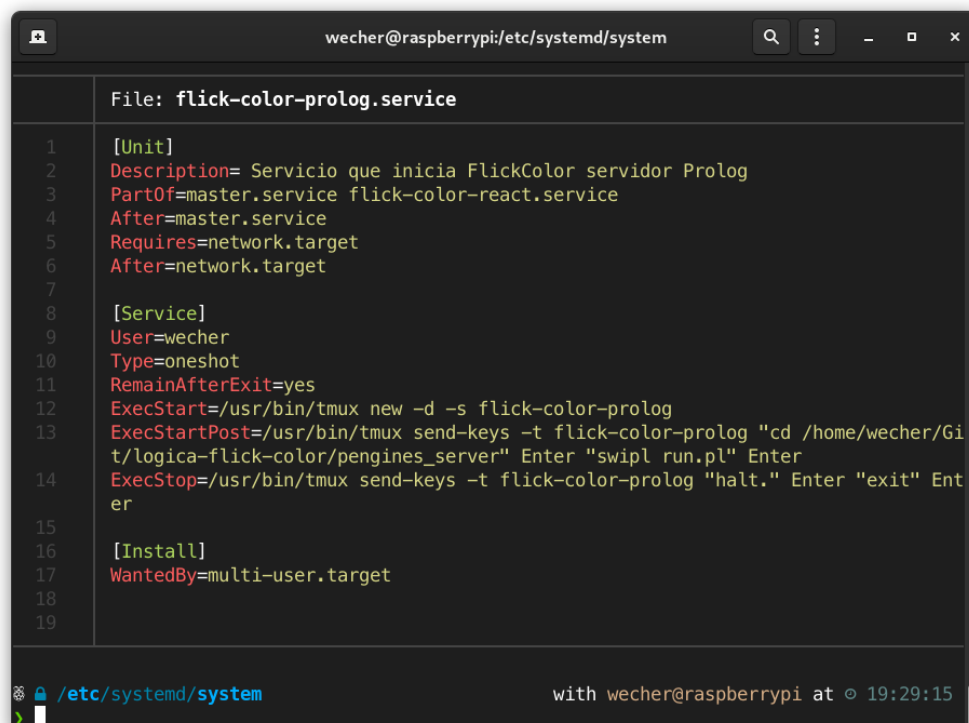
```
wecher@raspberrypi/etc/systemd/system
with wecher@raspberrypi at 19:28:34
> cat flick-color-react.service

File: flick-color-react.service

1  [Unit]
2  Description= Servicio que inicia FlickColor servidor React
3  PartOf=master.service
4  After=master.service network.target
5  Requires=network.target
6  Wants=flick-color-prolog.service
7
8  [Service]
9  User=wecher
10 Type=oneshot
11 RemainAfterExit=yes
12 ExecStart=/usr/bin/tmux new -d -s flick-color-react
13 ExecStartPost=/usr/bin/tmux send-keys -t flick-color-react "cd /home/wecher/Git
14 /logica-flick-color/" Enter "npm start" Enter
15 ExecStop=/usr/bin/tmux send-keys -t flick-color-react "C-c" "exit" Enter
16
17 [Install]
18 WantedBy=multi-user.target

wecher@raspberrypi/etc/systemd/system
with wecher@raspberrypi at 19:28:42
>
```

A continuación mostramos el servicio que levanta el servidor de React.

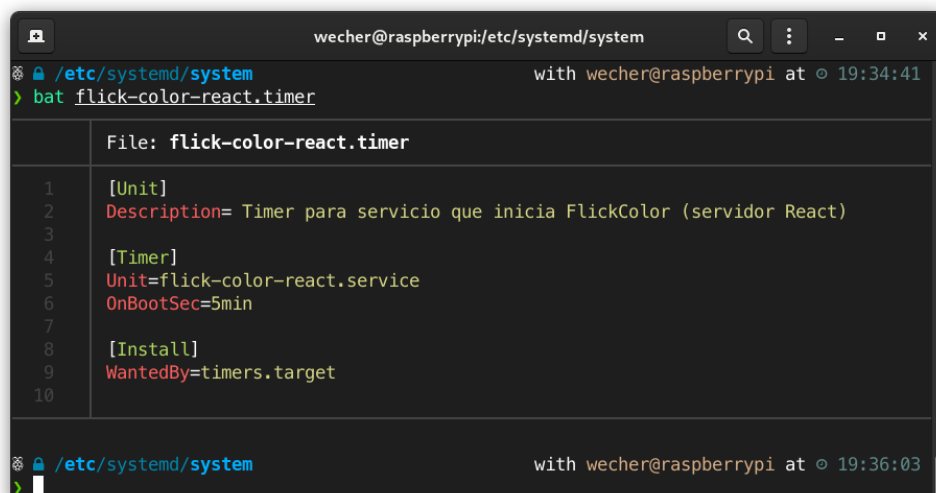


```
wecher@raspberrypi:/etc/systemd/system
File: flick-color-prolog.service
1 [Unit]
2 Description= Servicio que inicia FlickColor servidor Prolog
3 PartOf=master.service flick-color-react.service
4 After=master.service
5 Requires=network.target
6 After=network.target
7
8 [Service]
9 User=wecher
10 Type=oneshot
11 RemainAfterExit=yes
12 ExecStart=/usr/bin/tmux new -d -s flick-color-prolog
13 ExecStartPost=/usr/bin/tmux send-keys -t flick-color-prolog "cd /home/wecher/Gi
14 ExecStop=/usr/bin/tmux send-keys -t flick-color-prolog "halt." Enter "exit" Ent
15 er
16
17 [Install]
18 WantedBy=multi-user.target
19
20 /etc/systemd/system with wecher@raspberrypi at 19:29:15
```

Se puede apreciar que la variable “ExecStart” ejecuta una herramienta llamada [Tmux](#), ésto es para poder levantar cada servidor en una sesión de terminal diferente.

Por lo tanto, en cualquier momento podemos entrar a ver cualquiera de las “2 terminales” que están ejecutando “en segundo plano” el servidor de React y Prolog.

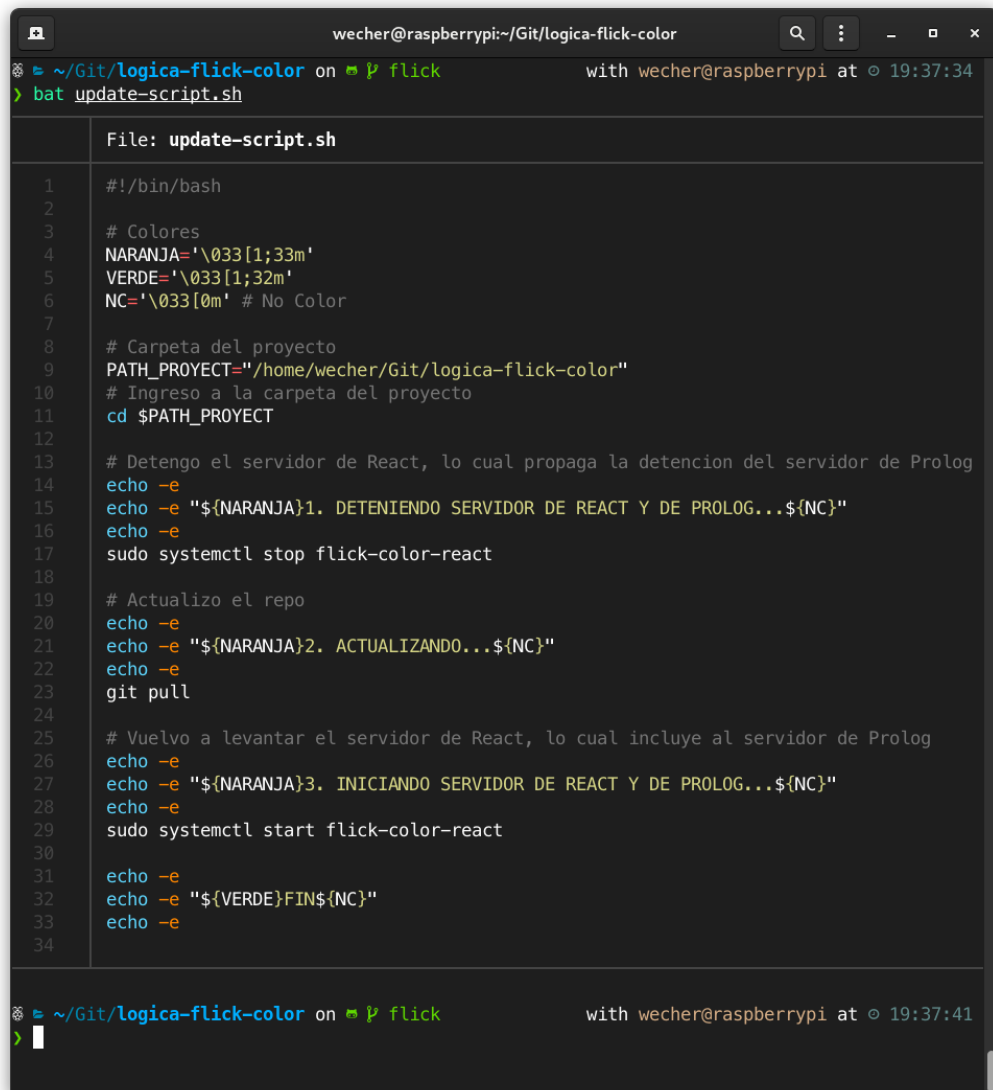
En caso de que haya un corte de luz, o que por algún motivo la raspi sea reiniciada, el sistema volverá a levantar ambos servidores automáticamente. Ésto lo logramos utilizando un “timer”, el cual se muestra a continuación.



```
wecher@raspberrypi:/etc/systemd/system
21 /etc/systemd/system with wecher@raspberrypi at 19:34:41
> bat flick-color-react.timer
File: flick-color-react.timer
1 [Unit]
2 Description= Timer para servicio que inicia FlickColor (servidor React)
3
4 [Timer]
5 Unit=flick-color-react.service
6 OnBootSec=5min
7
8 [Install]
9 WantedBy=timers.target
10
22 /etc/systemd/system with wecher@raspberrypi at 19:36:03
```

El proyecto lo tenemos alojado en Git, de ésta forma podemos actualizar fácilmente el juego que está ejecutándose en la raspi. Para ello, hicimos un script en [Bash](#) que automatiza el siguiente proceso en la raspi:

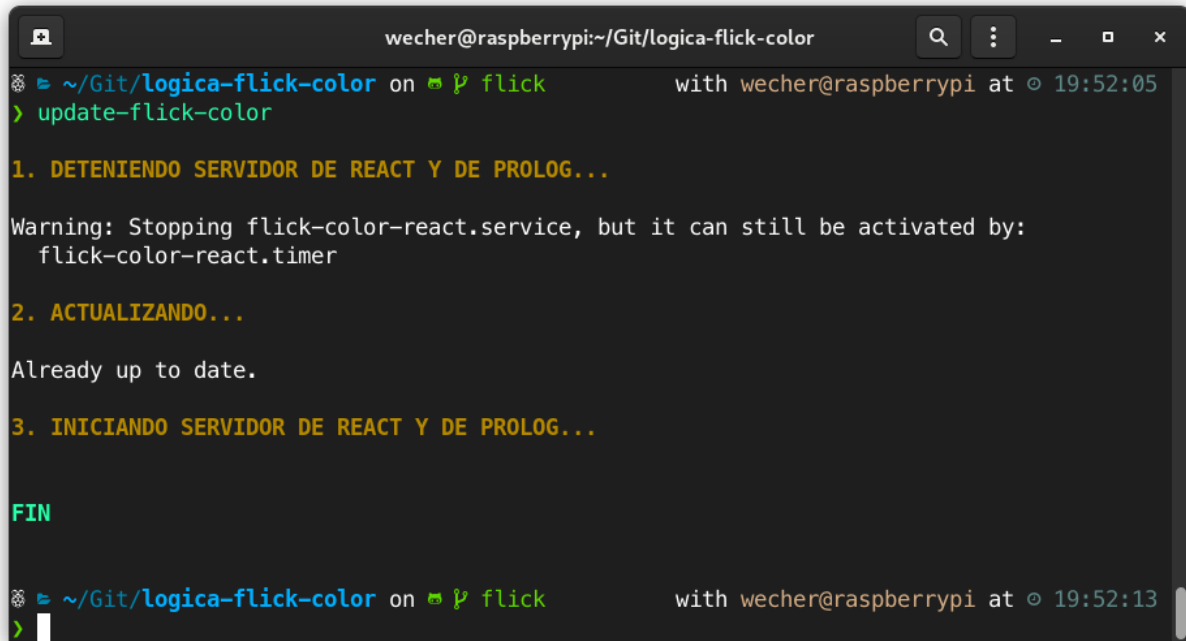
1. Detener el servidor de React y el servidor de Prolog
2. Actualizar los cambios desde git
3. Levantar nuevamente ambos servidores



```
wecher@raspberrypi:~/Git/logica-flick-color
~/Git/logica-flick-color on flick with wecher@raspberrypi at 19:37:34
> bat update-script.sh

File: update-script.sh
1  #!/bin/bash
2
3  # Colores
4  NARANJA='\033[1;33m'
5  VERDE='\033[1;32m'
6  NC='\033[0m' # No Color
7
8  # Carpeta del proyecto
9  PATH_PROYECT="/home/wecher/Git/logica-flick-color"
10 # Ingreso a la carpeta del proyecto
11 cd $PATH_PROYECT
12
13 # Detengo el servidor de React, lo cual propaga la detencion del servidor de Prolog
14 echo -e
15 echo -e "${NARANJA}1. DETENIENDO SERVIDOR DE REACT Y DE PROLOG...${NC}"
16 echo -e
17 sudo systemctl stop flick-color-react
18
19 # Actualizo el repo
20 echo -e
21 echo -e "${NARANJA}2. ACTUALIZANDO...${NC}"
22 echo -e
23 git pull
24
25 # Vuelvo a levantar el servidor de React, lo cual incluye al servidor de Prolog
26 echo -e
27 echo -e "${NARANJA}3. INICIANDO SERVIDOR DE REACT Y DE PROLOG...${NC}"
28 echo -e
29 sudo systemctl start flick-color-react
30
31 echo -e
32 echo -e "${VERDE}FIN${NC}"
33 echo -e
34
~/Git/logica-flick-color on flick with wecher@raspberrypi at 19:37:41
> 
```

Al ejecutar el script anterior, vemos el siguiente resultado:



```
wecher@raspberrypi:~/Git/logica-flick-color
~/Git/logica-flick-color on flick with wecher@raspberrypi at 19:52:05
> update-flick-color

1. DETENIENDO SERVIDOR DE REACT Y DE PROLOG...

Warning: Stopping flick-color-react.service, but it can still be activated by:
flick-color-react.timer

2. ACTUALIZANDO...

Already up to date.

3. INICIANDO SERVIDOR DE REACT Y DE PROLOG...

FIN

~/Git/logica-flick-color on flick with wecher@raspberrypi at 19:52:13
>
```

De ésta forma, basta con ejecutar dicho script, el juego ya es actualizado a la última versión, en cuestión de segundos.



Frontend

Modificamos totalmente el frontend con respecto a la primera entrega. Le cambiamos el fondo haciéndolo degradado en tonos cálidos. Agregamos un header indicando el nombre del juego y nuestros nombres. Debajo un footer indicando “Trabajo realizado para la materia lógica para ciencias de la computación”.

Separamos cada sección del juego con bordes, haciéndolo más claro visualmente, y utilizamos la librería [sweetalert2](#) para reemplazar los avisos por carteles con animaciones, haciendo que la interacción con el usuario sea más clara y con un diseño más atractivo.

Records de turnos

Para ésta segunda etapa, decidimos realizar un panel en el que figura el récord de cada usuario.

El nombre se obtiene antes de empezar a jugar. El récord de turnos es la cantidad total de turnos que tuvo que jugar el usuario para poder ganar el juego.

Recordar que el nombre debe empezar con letra minúscula (ver manual de usuario).

A continuación mostramos imágenes del código en React para modelar ésta función.

```
1  nuevoRecord(){
2    const queryS = "newRecord(" + this.state.nombreJugador + "," + this.state.turns + ", AllRecords)"
3    this.setState({
4      waiting: true
5    });
6    this.pengine.query(queryS, (success, response) => {
7      if (success) {
8        this.setState({
9          waiting: false,
10        });
11      } else {
12        this.setState({
13          waiting: false
14        });
15      }
16    });
17  }
18
19  getRecords(){
20    const queryS = "getRecords(Records)";
21    this.setState({
22      waiting: true
23    });
24    this.pengine.query(queryS, (success, response) => {
25      if (success) {
26        this.setState({
27          records: response['Records'],
28          waiting: false,
29        })
30      }
31    })
32  }
33
34  async getNombre(){
35    const { value: nombre } = await Swal.fire({
36      title: '¡Bienvenido a Flick Color!',
37      input: 'text',
38      inputLabel: 'Ingrese su nombre para poder registrar su puntuación',
39    })
40    this.setState({
41      nombreJugador: nombre
42    });
43  }
```


A continuación mostramos imágenes del código en Prolog para modelar ésta función.

```
1  getRecords(RecordsOrdenados):-
2      % busco todos los ganadores
3      findall([Nick,Turnos],
4              ganador(Nick,Turnos),
5              RecordsDesordenados),
6      % ordeno los ganadores segun sus turnos (de menor a mayor)
7      sort(2, @<, RecordsDesordenados, RecordsOrdenados).
8
9  % Caso 1: el usuario no existe en la base de datos, por lo tanto registramos su puntuacion
10 newRecord(Nick,Turnos,NewRecords):-
11     not(ganador(Nick,_)),
12     assert(ganador(Nick,Turnos)),
13     getRecords(NewRecords),
14     !.
15 % Caso 2: el usuario ya existe en la base de datos
16 newRecord(Nick,Turnos,NewRecords):-
17     % buscamos su puntuacion
18     ganador(Nick,OldTurnos),
19     % solo registramos su puntuacion si hizo un record
20     Turnos < OldTurnos,
21     !,
22     retract(ganador(Nick,OldTurnos)),
23     assert(ganador(Nick,Turnos)),
24     getRecords(NewRecords).
25 % Caso 3: siempre retorna verdadero
26 newRecord(_,_,Records):- getRecords(Records).
```

Como se puede apreciar, cada nombre va a tener solamente 1 record. Por ejemplo, si el jugador de nombre “mati” juega 20 veces siempre con ese nombre, entonces en la tabla de records solamente va a aparecer 1 sola vez, con el nombre “mati” y con la menor cantidad de turnos en la que pudo ganar el juego.



CONCLUSIÓN

Para ésta segunda etapa, nos sentimos más afianzados con el lenguaje de programación Prolog. También tuvimos más tiempo para investigar sobre React, y así mejorar el frontend del juego.

Aprovechamos todas las consultas prácticas y de laboratorio brindadas por la cátedra.

Además, pudimos realizar funciones extras, que detallamos a lo largo del informe.

Para ésta segunda entrega, seguimos trabajando en conjunto manteniendo una buena organización y comunicación entre ambos, mayormente de forma virtual por distintos medios digitales.

El proyecto nos pareció muy interesante de desarrollar, ya que tuvimos la suficiente libertad para agregar funciones extras, tanto de backend como del frontend.