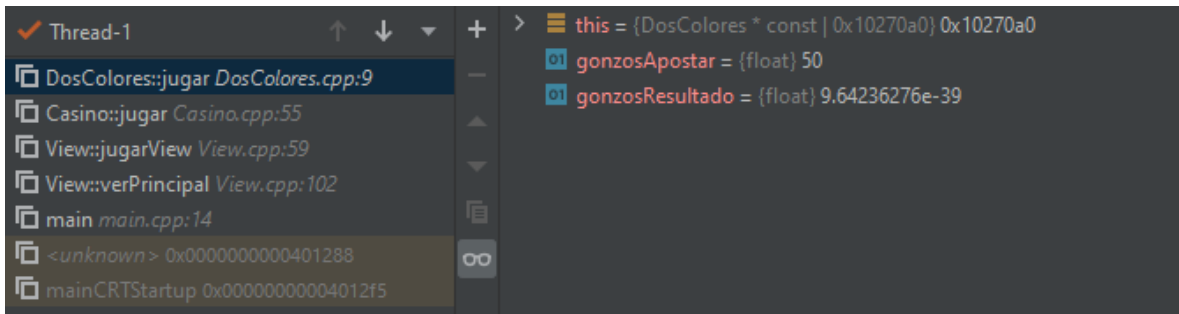
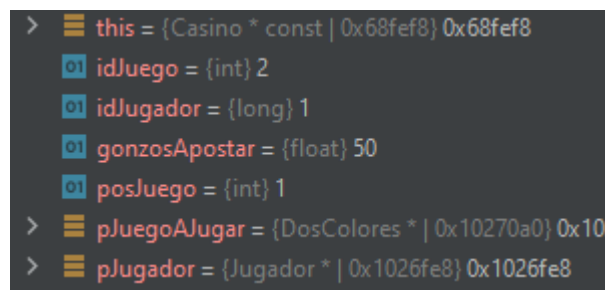


1. Agregue un breakpoint al programa de casino en el método jugar de la clase DosColores. Pruebe la ejecución hasta que llegue al punto del breakpoint y su programa se interrumpa. En ese punto tome una foto de los frames disponibles e indique:
 - a. Cuál fue el método que inició toda la ejecución
 - b. Explore que elementos tiene la variable `this` en el frame DosColores, en el frameW Casino y en el frameW jugarView. Tome una foto que evidencie los valores de las variables de algunos de estos frames.

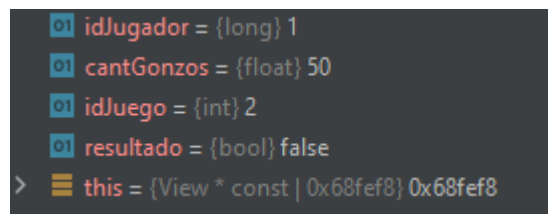


- a. La ejecución empieza desde el método jugar del View cuando se llama al jugador específico (quién y cuánto apostará) y el jugar de Casino para escoger un juego.
- b. **FrameW DOSCOLORES:** Imagen superior. Tiene los gonzos a apostar y el resultado que es el valor flotante que inicializa Dos Colores.

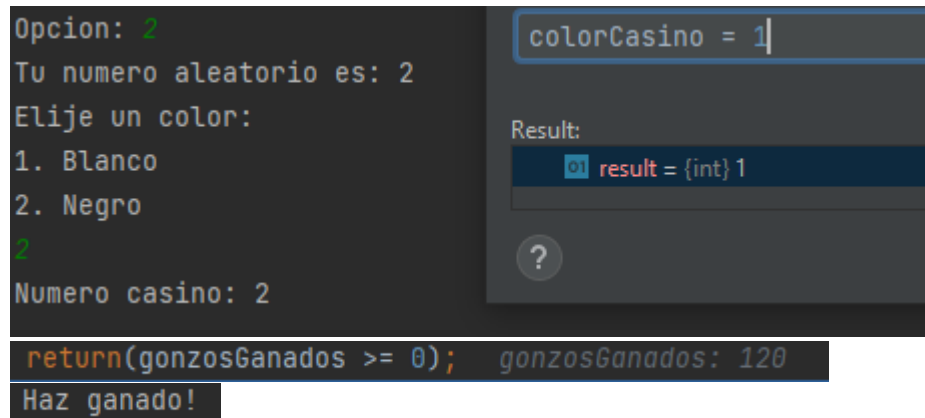
FrameW CASINO: ID del juego seleccionado (como se inicializó en el casino) y del jugador, junto con su apuesta. PosJuego que corrige la ID del juego según la selección del usuario y los punteros que indican el jugador y el juego.



FrameW VIEW: Contiene gran parte de los elementos que fueron enviados a Casino para iniciar a jugar (siendo las IDs) y el booleano que indica si gana o pierde (que será cambiado dependiendo de lo que ocurra en el frame del juego seleccionado)



2. **Set variable values:** en el mismo breakpoint, una vez se han generado las variables aleatorias para el casino y para el jugador ajústelas directamente en el debug para asegurarse de que el jugador gane.



```
Opcion: 2
Tu numero aleatorio es: 2
Elije un color:
1. Blanco
2. Negro
2
Numero casino: 2

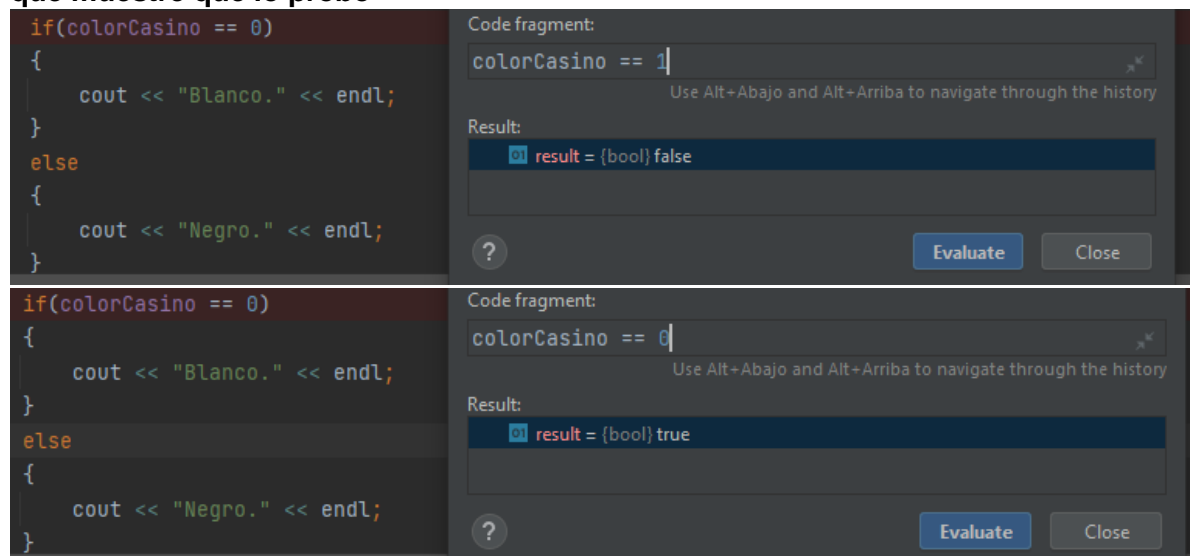
return(gonzosGanados >= 0); gonzosGanados: 120
Haz ganado!
```

colorCasino = 1

Result:

01 result = {int} 1

3. Cree un breakpoint que se lance después de que ocurran cierto número de repeticiones.
4. BreakPoint condicional: incorpore un breakpoint condicional en el código fuente. Explique en qué funcionalidad lo incorporó. Tome una foto que muestre que el programa se interrumpió exitosamente y muestre el estado de las variables cuando se interrumpió la ejecución.
5. Pruebe el evaluador de expresiones con alguna expresión. Tome una foto que muestre que lo probó



```
if(colorCasino == 0)
{
    cout << "Blanco." << endl;
}
else
{
    cout << "Negro." << endl;
}
```

Code fragment:

colorCasino == 1

Result:

01 result = {bool} false

Evaluate Close

```
if(colorCasino == 0)
{
    cout << "Blanco." << endl;
}
else
{
    cout << "Negro." << endl;
}
```

Code fragment:

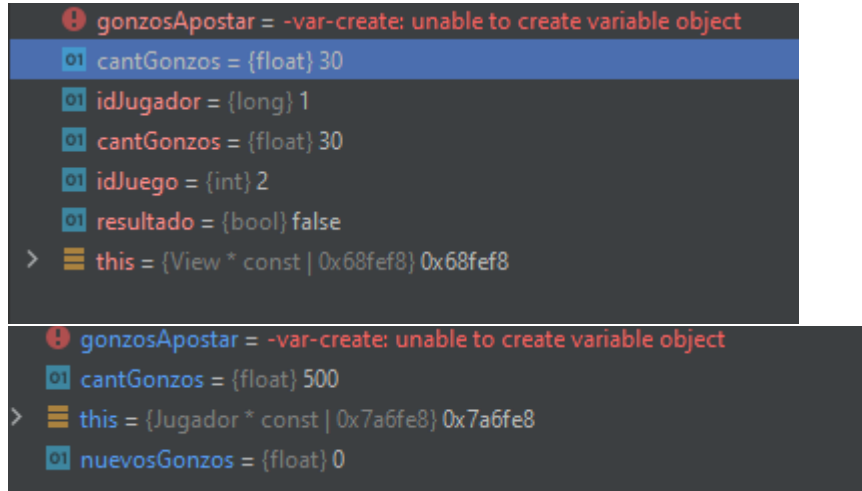
colorCasino == 0

Result:

01 result = {bool} true

Evaluate Close

6. Agregue un watch para observar alguna de las variables definidas a lo largo del casino.



The screenshot shows a debugger's 'Watches' window. It contains two entries: 'cantGonzos' with a value of {float} 30, and 'nuevosGonzos' with a value of {float} 0. Above these, there is a red error message: 'gonzosApostar = -var-create: unable to create variable object'. The debugger interface is dark-themed with blue and yellow highlights.

Un watch en la cantidad de gonzos del jugador.

7. Explique: qué diferencia encontró entre la funcionalidad step over (F8) y la funcionalidad step into (F7). Explique con un ejemplo de las funcionalidades del casino.

StepInto: Cuando la siguiente instrucción a ejecutar llega a una llamada a un método, no ejecuta el método, sino que ejecuta la primera línea de ese método y se detiene.

Step Over: Cuando la siguiente instrucción a ejecutar llega a una llamada a un método, ejecuta el método como un todo.