A chess clock is a clock which has two adjacent clocks(timers), with two buttons, which stop one clock and start the other, so the clocks never run simultaneously. So to make this I will use two buttons and two timers.

First we configure two timers: Timer 0 and Timer 1. We enable interrupts on both of them as well as resetting them and stopping if they are working. After this we set the timer(match register) with some amount of time, in case of my program I set it to 15 seconds, because the recording had to be smaller than 60 seconds. After this we set it up to do IRQ's (Interrupt requests). We set it with low priority, upload whatever address the interrupt came from and enable timer interrupt channels. After this is done we configure two buttons: Button 1 and Button 2. We set them so they work as external interrupts and make them work only when they are pressed and don't work when they are released(working on rising-edge). After this we do the same thing we did to timers we set them up so the interrupt channels work and so on.

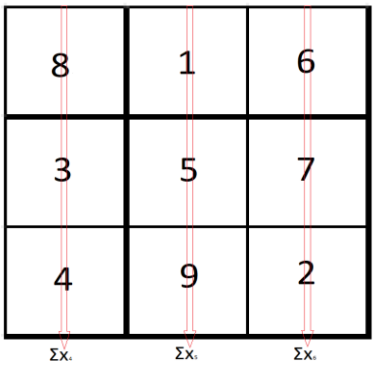
When an IRQ exception is raised, we go to the IRQ handler which manually redirects us to either button 1, button 2 or timer 0, timer 1 handlers. It can do this because we configured the VIC vectors before, having address loaded to every button or timer channel. Timer 0 and timer 1 do the same thing, when any of the two timers hits the 15 second mark it raises an IRQ exception which then redirects it to any of the timer handlers, which do the same thing. They disable the buttons so after the time is over for one you can't press the button to start the other time. When the button 1 interrupt is raised it goes to Button 1 handler where it stops the timer 0 and starts the timer 1. Button 2 is very similar to button 1 with only a slight difference, it stops the timer 1 and starts the timer 0. As well as both buttons not working when any of the times is over.

For example if you press button when there is 3 seconds in first timer and 0 in second, it stops timer 0 at 3 seconds and starts incrementing timer 1 from 0 seconds until it hits 15 seconds(in case of my program) or until button 2 is pressed, where it will stop timer 1 and start timer 0 again.

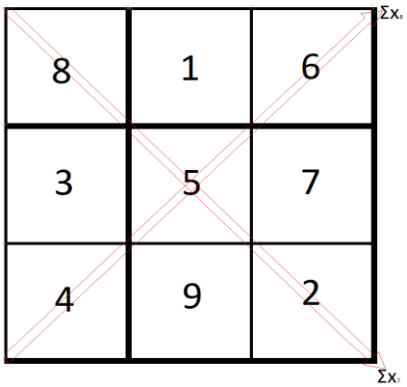
Source:
https://en.wikipedia.org/wiki/Chess_clock

A magic square is a square array of numbers consisting only from positive integers arranged in a way that the sum of the numbers in any horizontal, vertical, or main diagonal line is always the same number. So to check if the square two-dimensional array in the memory is magic square I would:
-Check if rows are magic

-Check if columns are magic



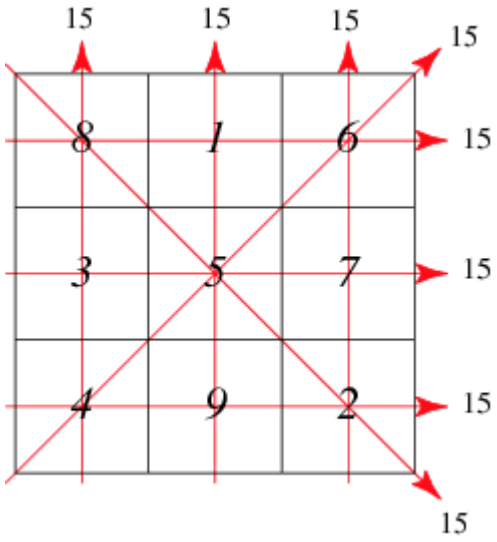
-Check if main diagonals are magic.



I decomposed the program into different subroutines. The constants are R0-address of the array, R1-size of array, R-3 is set to 1 by default. I had a subroutine “magicSumNumber” which takes in the address of the array and its size and finds what the sum of first row numbers is (I do this because if all the rows sums and column sums and diagonal sums are the same so they should equal to one another either way, and there is no need to find all the numbers separately and check them against all the

values individually when you can get one number and check all the other numbers against it). Then it calls the first subroutine “checkIfRowsMagic” which takes in the address of the array and its size also takes the number found previously lets call it magic number. After this it goes through each row of the two-dimensional array, adds the numbers on each row, checks if it’s the same as a magic number if it isn’t it sets the number in register R3 to 0, and when it goes back to where it was called it skips to the end of program. If it is the same, then the next subroutine is called “checkIfColumnsMagic” which has the same inputs as the previous subroutine. After this it goes through each column of the two-dimensional array, adds the numbers on each column, checks if it’s the same as a magic number and does the same thing as the previous subroutine. Again if R3 still is 1 then the next subroutine is called “checkIfDiagonal1Magic” has the same inputs as other subroutines, but it checks through the center diagonal from left to right, from top to bottom. Does the same things as previous (setting R3 - 0 or 1). If R3 is still 1 goes into the last subroutine which has the same inputs it checks if the center diagonal from right to left, from top to bottom number sum is the same as magic square, has the same outputs as previous ones. So if after this subroutine is over and R3 is still 1 that means the square array of numbers is a magic square.

So this is an example of the square matrix being a magic square, as it’s individual row sums, individual column sums and individual diagonal sums are the same. 8+1+6=15 as is 8+3+4=15 as is 8+5+2=15 and so on, we can see the sum of all horizontal, vertical, or main diagonal lines are the same 15.



My pseudo-code is:
public static void main(String[] args)

```
{
	int arr1Size = 3;
	int [ ] [ ] arr1 = { {8, 1, 6},
			     {3, 5, 7},
			     {4, 9 ,2} };
	boolean isMagicSquare = true;
	isMagicSquare  = magicSumNumber(arr1, arr1Size)
	if(isMagicSquare == true)
	{
		System.out.print(“The square array is a magic square”);
	}
	else if(isMagicSquare == false)
	{
		System.out.print(“The square array is not a magic square”);
	}
}

public boolean magicSumNumber(int [ ] [ ] arr1, int arr1Size)
{
	boolean isMagic = true;
	int magicNumber = 0;
	for(int j = 0; j < arr1Size; j++)
	{
		magicNumber = magicNumber + arr1[ 0 ][ j ]
	}

	if(checkIfRowsMagic(arr1, arr1Size, magicNumber)==false)
	{
		isMagic = false;
	}

	if(checkIfColumnsMagic(arr1, arr1Size, magicNumber)==false)
	{
		isMagic = false;
	}

	if(checkIfDiagonal1Magic(arr1, arr1Size, magicNumber)==false)
	{
		isMagic = false;
	}
```

```
	if(checkIfDiagonal2Magic(arr1, arr1Size, magicNumber)==false)
	{
		isMagic = false;
	}

	return isMagic;
}

public boolean checkIfRowsMagic(int [ ][ ] arr1, int arr1Size, int magicNumber)
{
	boolean isRowsMagic = true;
	for(int i = 0; i < arr1Size; i++)
	{
		int magicRowNumber = 0;
		if(isRowsMagic == true)
		{
			for(int j = 0; j < arr1Size; j++)
			{
				magicRowNumber = magicRowNumber + arr1 [ i ] [ j ];
			}
			if(magicRowNumber != magicNumber)
			{
				isRowsMagic = false;
			}
		}
	}
return isRowsMagic;
}

public boolean checkIfColumnsMagic(int [ ][ ] arr1, int arr1Size, int magicNumber)
{
	boolean isColumnsMagic = true;
	for(int j = 0; j < arr1Size; j++)
	{
		int magicColumnNumber= 0;
		if(isColumnsMagic  == true)
		{
			for(int i = 0; i < arr1Size; i++)
			{
```

```
                        magicColumnNumber = magicColumnNumber + arr1 [ i ] [ j ];
                    }
                    if(magicColumnNumber != magicNumber)
                    {
                        isColumnsMagic = false;
                    }
                }
        }
return isColumnsMagic;
}

public boolean checkIfDiagonal1Magic(int [ ][ ] arr1, int arr1Size, int magicNumber)
{
        boolean isDiagonal1Magic = true;
        for(int j = 0, i = 0; i < arr1Size; j++, i++)
        {
                magicDiagonal1Number = magicColumnNumber + arr1 [ i ] [ j ];
        }
        if(magicDiagonal1Number !=  magicNumber)
        {
                isDiagonal1Magic = false;
        }
return isDiagonal1Magic;
}

public boolean checkIfDiagonal2Magic(int [ ][ ] arr1, int arr1Size, int magicNumber)
{
        boolean isDiagonal2Magic = true;
        for(int i = 0, j = arr1Size; i < arr1Size; j--, i++)
        {
                magicDiagonal2Number = magicColumnNumber + arr1 [ i ] [ j ];
        }
        if(magicDiagonal2Number !=  magicNumber)
        {
                isDiagonal2Magic = false;
        }
return isDiagonal2Magic;
}
```

Sources:
https://mathworld.wolfram.com/MagicSquare.html
https://en.wikipedia.org/wiki/Magic_square

My program is fairly efficient, it first finds a magic number(a number which sum of first row equals to) and then checks against it other sums of rows, columns and main diagonals it gets. Also if one of them does not equal to a magic number it skips the other processes of subroutine as it is evident that square array is not magic square. The program is divided into different subroutines which makes it easy to follow. Big thing to improve on would be that if the sum of diagonals is not equal to the magic number it has to go through the rows and columns and only after going through those subroutines will it go through the diagonal subroutine. Also would want to improve on diagonal subroutine as there are two of them for each diagonal, could improve on it by turning it into one. I used a fairly small number of registers in the subroutines and the program overall which is a good thing. Overall this program is fairly efficient and commented well, so it is pretty easy to follow.