Tecnológico de Monterrey

OBLITERATION

A two-player Java game with concurrency and sockets

Programming Languages - Final Project

Professor: Benjamín Valdés Aguirre

24/05/2020

Oswaldo David García Rodríguez - A01206725

CONTENT

1. CONTEXT

Obliteration is a game that consists on a grid of hexagons, where each player must convert as much tiles as they can to a color they have assigned.

It's like classic board games as chess, checkers, solitaire or Chinese checkers in the part of strategy. But also depends on random elements like domino or UNO.

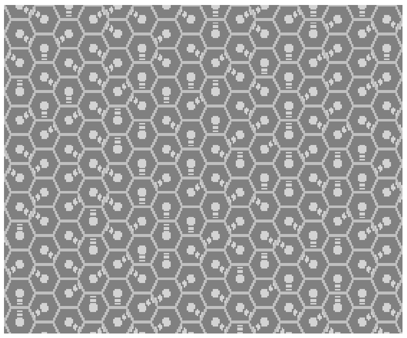


Figure 1.1 Example of a grid. Each hexagon is 'connected' to another by pointing at a random direction.

The problem with the game is that, although the rules are as simple as checkers or Chinese checkers, the accommodation of the grid is hard to setup physically, considering each hexagon have a direction and can be 'rotated'. Also, when a player converts an hexagon to their color, all other hexagons connected to the converted will have to be also recolored. During the game, this will be tedious. Those are reasons why a game like this is not practical to be played in a physical version.

The purpose of the project is to provide this 'physically impractical' game using a graphical interface. Also, this solution will be implemented using sockets, so both players in a local network can execute the game on different computers.

Note: The original game name is Corruption, that can be played on https://www.newgrounds.com/portal/view/714303. In this version, there are two different game modes. Domination mode finishes when a player converts at least 70% of the hexagons in the grid. Obliteration mode finishes when a player converts all enemy hexagons.

2. RULES OF THE GAME

GRID SETUP

Each hexagon is a tile that has a rotation. Each one of those must be placed on the grid with a randomly chosen rotation.

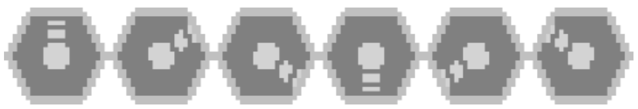


Figure 2.1 All possible rotations of an hexagon. For practical purposes, those are numbered from 0 to 5. Notate those rotations have a clockwise direction.

This game is for 2 players and each one has a different color to play. The upper left corner hexagon is for player 1 and has rotation 1. The lower right corner hexagon is for player 2 and has rotation 4.

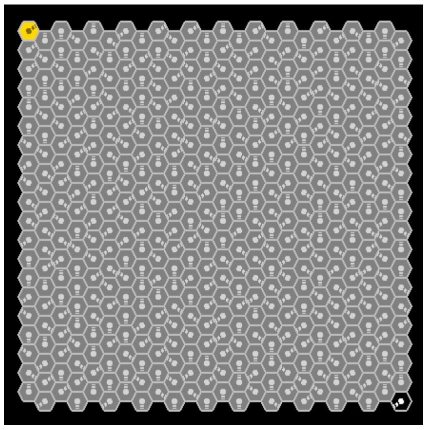


Figure 2.2 Example of a grid. Player 1's hexagon is yellow and has rotation number 1. Player 2's hexagon is black and has rotation 4.

PLAY SEQUENCE

Player 1 starts by 'rotating' their hexagon. In this case, their hexagon goes from rotation 1 to 2. Please note it's not permitted to rotate counterclockwise (for example, from 5 to 4).

After this, the rotated hexagon is going to point to another, which will be converted to the player's color. Consider that if the converted hexagon is pointing to another, it must be also colored. Finally, all hexagons pointing to the converted ones will be also repainted. Repeat this process until all connected hexagons are repainted.

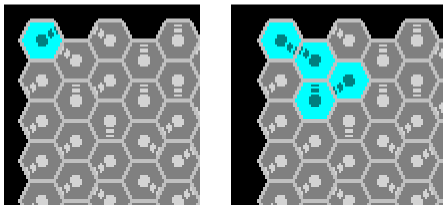


Figure 2.3. After rotating the player 1's hexagon from position 1 to 2, all other connected are converted to the player's color.

'Connected' is when an hexagon is pointing to another. It doesn't matter if the other hexagon it's not pointing back to it.

When this process finishes, then it's the player 2's turn. After this, player 1 will have more hexagons of their color. He or she can rotate any of them.

It may happen that the player's rotation is going to point outside the grid or to an hexagon that is already colored to the player's color. In those cases, there is not going to be any conversions to make.

Also, a player's rotation could point to opponent's hexagons. Those must be converted too.

WIN CONDITION

When a player converts all opponent hexagons, it means that, in the next turn, that opponent won't be able to play. That means the game is over. The player who still have hexagons of their color is the winner.

In this game, there is not possibility of a tie.

3. SOLUTION PROPOSED

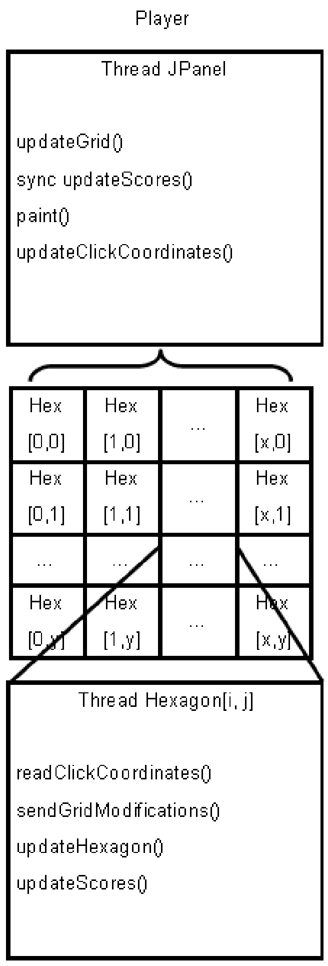


Figure 3.1 Basic architecture of player's JPanel

Considering that sometimes a lot of hexagons must be converted after a rotation, during a game with a physical version, this process will be very tedious as explained before. Another important problem is that keeping track of all the conversions that must be made each turn would get confusing for both players. So, it's probable they make a lot of mistakes, making the game hard, even with these simple rules.

That's why the proposal of this project is to program the game to make the conversion process automatically, so that the players just need to focus on their strategies.

In order to do this, it is required a graphical interface. Also, the hexagons should be changed using mouse clicks. For that case, each hexagon should work as a thread that is constantly listening to the position of the last click.

The graphical interface (which from now on will be referred to as JPanel), has the responsibility to repaint the whole board, which includes the grid and scores. Also, it will send to the other player the information about their last movement so the opponent may update their own board.

Scores are calculated as a percentage. The score of each player is equal to their colored hexagons divided by hexagons converted by both players. But that will be just a visual implementation since the winning condition will be converting all enemy tiles.

On the other hand, the hexagons will send to the JPanel all information about its new rotation (if applies) and all hexagons that should change thanks to the conversions. Notice that the hexagon won't directly repaint the grid, it's just sending the list of changes to apply so JPanel update them and notify the rest of hexagons. Once they heard they were modified, they update the scores using the synchronized void that it's on JPanel.

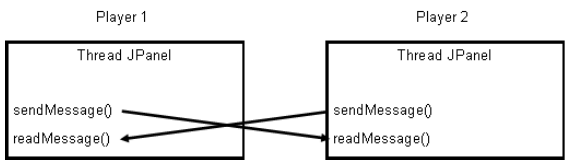


Figure 3.2. Communication between players through sockets

The following flow diagram explains how classes JPanel and Hexagon interact during the game. For simplicity, it doesn't explain in a deep detail how the scores are updated. It's also important to mention that its purpose is to explain the logic of the game, so it neither makes emphasis on the sockets' functionality, so it's only mentioned when the information is sent to the other player to update their board.

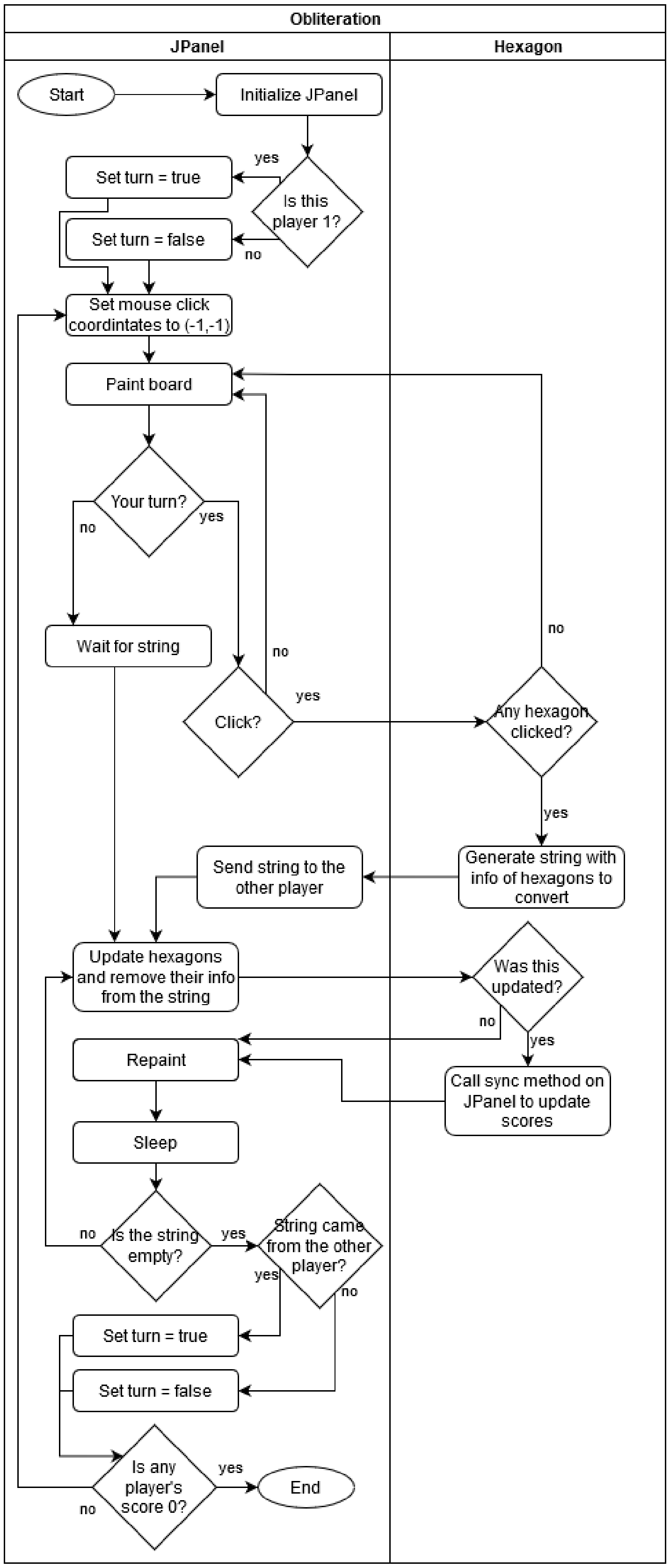


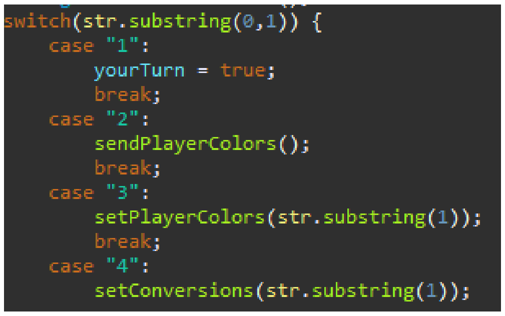
Figure 3.3 Flow diagram of the game's logic.

4. IMPLEMENTATION

When the game is opened, it will show a console to ask the user for the IP and the port number to connect both players (except for the 'local' version, which will automatically connect to localhost:22222 without using a command prompt).

When the address is entered, the game will try to connect to an existing game in that IP. If it's already taken by two players, is in use or is invalid, it will show a message indicating that and ends execution. If there's a game with only one player, it will connect as player 2. If the address is available and there's no game started, it will show a message saying it was unable to connect to a game on that address, so it will start the server being player 1.

Player 1 then creates the grid and players' colors. However, it will show a message indicating that is waiting for player 2, so the board won't be shown yet. When player 2 starts the game, player 1 will send him a string starting with a number 4 followed by the number's rotation and color of each hexagon. Then it will send another string that starts with number 3 followed with the colors of the players.



```
switch(str.substring(0,1)) {
    case "1":
        yourTurn = true;
        break;
    case "2":
        sendPlayerColors();
        break;
    case "3":
        setPlayerColors(str.substring(1));
        break;
    case "4":
        setConversions(str.substring(1));
```

Figure 4.1 Menu that identifies what type of message was received from the other player. It reads the first character of the string.

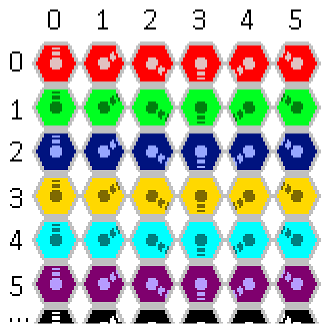


Figure 4.2 Sprites sheet. In horizontal, the numbers to identify rotations in game. In vertical, the numbers of the colors.

After sending those strings, player 1 will have a screen which says 'CLICK TO START'. If the player does that, the board can be seen, but can't play until player 2 clicks their screen too. This is to prevent player 1 to play until player 2 confirms their participation.

Player 1 starts. He or she only may click on the hexagon of their color. As it can be seen in figure 2.2, the first click will make the only colored hexagon to point to another, so that means their first movement is guaranteed to convert at least one tile (the same for player 2).

The following figure shows an example of a conversion made after a rotation. It can be noticed that, in the 5th image, the hexagon marked with "…" was not converted although it was pointing to a recently recolored tile, but after step **7** it's reached.

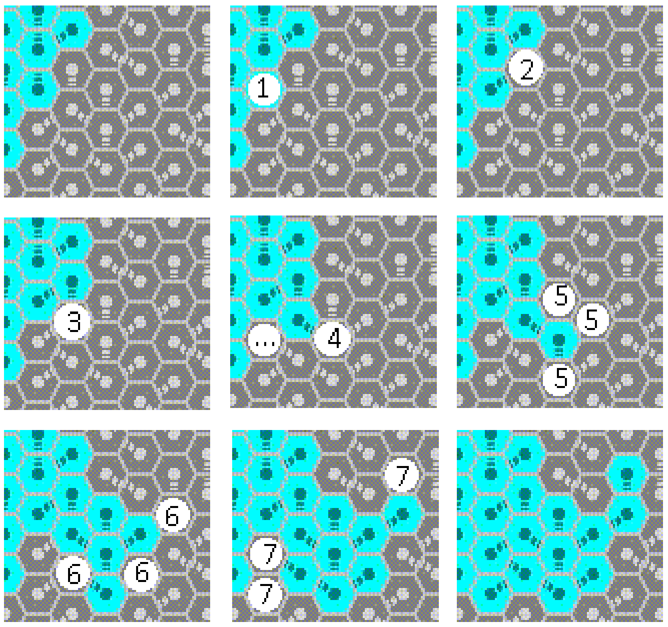


Figure 4.3 Sequence of the hexagon's conversions after one rotation. Numbers indicate the order in which the tiles will be repainted. It has an example of an "infinite conversion loop".

Why did this happen? Notice that hexagon marked with **3** converted the marked with **4**, then it converted the lower **5**, then the left **6**, and **6** converted the upper **7**, which previously was marked with '…'. It means that hexagon had **2** ways to be converted, but the algorithm of conversions first detected the conversion **7** instead '…'.

It is supposed that, logically, the '…' conversion had to be detected first, but that didn't happen. The reason is that, after the rotation was made, the hexagon rotated executed an algorithm to detect the order the conversions. If the first hexagon converted is pointing outside the board or a current player's hexagon, then it will check the hexagons that are pointing to it to convert them too. If it's not pointing outside the board or to a current player's hexagon, that means it's pointing to a tile that must be also converted, so the algorithm recursively starts again on the converted tile. With that algorithm, the hexagon marked with '…' was detected first in the **7**th level of conversion instead of the **4**th.

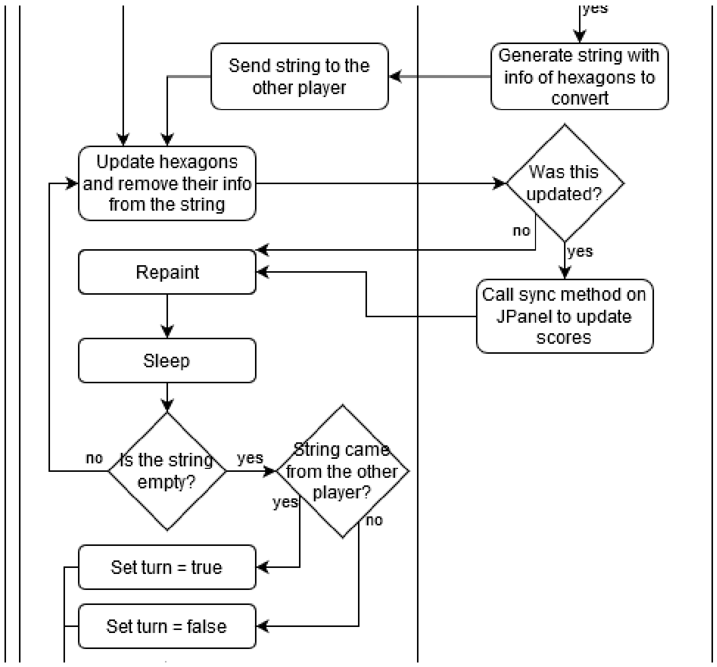
This may happen sometimes but helps to prevent hexagons to reconvert tiles detected before, also from generating an 'infinite conversion loop'. However, all of them are converted at the end, so this is just a visual problem that, in fact, it's almost never noticed by the human eye.

The algorithm creates an Array List of class String in order to add the information of the hexagons to convert in the corresponding iterations. Looking at the figure 4.3, in the position 0 of the Array List, it is added the information of the rotated hexagon (marked with number 1). In position

4 of the Array List, for example, they is added the information of hexagons marked with 5.

The information of each hexagon consists in two digits for the horizontal position, then two for vertical position, two more for the color according to the sprite sheet and two extra digits for the rotation number.

After iterating through all hexagons to convert, the Array List is condensed in a single String object. Between each String from the original array, it will be placed a '++' in the new String so the program can identify when to pause from converting hexagons. It's important to mention that at the beginning of the String, it is added a number 4. That's in order to the menu in figure 4.1 identifies that the string represents a list of conversions. At the end, it is added a pair or characters to identify which player sent the conversions. If the receptor player is the same as the sender, it means it had their turn. Otherwise, the player knows the message came from the opponent, so it's the turn for this receptor player.


yes
Send string to the other player
Generate string with info of hexagons to convert
Update hexagons and remove their info from the string
Was this updated?
no
yes
Repaint
Call sync method on JPanel to update scores
Sleep
no
Is the string empty?
yes
String came from the other player?
yes
no
Set turn = true
Set turn = false
Figure 4.4 Part of the flow diagram that creates, sends and manipulates the String. Player sends it directly to itself using a single function. To send it to the opponent, it uses a socket.

The program converts hexagons reading each following 8 characters. It stops when finds a '++'. After updating, the hexagons converted call a sync function to update the score in JPanel. Then, the JPanel is repainted, sleeps for a moment and then checks if there still is information in the String. At the end, it identifies which player sent the String, so each player knows who has the next turn.

As mentioned before, hexagons have an X and Y position. Vertically, it can be easily seen how the columns are positioned. Horizontally, the rows are forming a zig-zag accommodation.

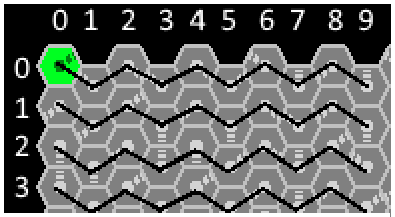


Figure 4.5 Position in X and Y in the grid. The black zig-zag lines indicate how the X positions are organized.

When hexagons are created, each one needs to know whose are their 'neighbors', that is, which hexagons are in direct touch with it.

Each one of the hexagons has an array of 6 pairs of coordinates to indicate the positions of their neighbors. The position in array of each pair also corresponds to the rotation number. For example, the pair in position [0] represents coordinates of the hexagon that is above (above is rotation 0). In position [3] are the coordinates of the lower hexagon (lower rotation is number 3).

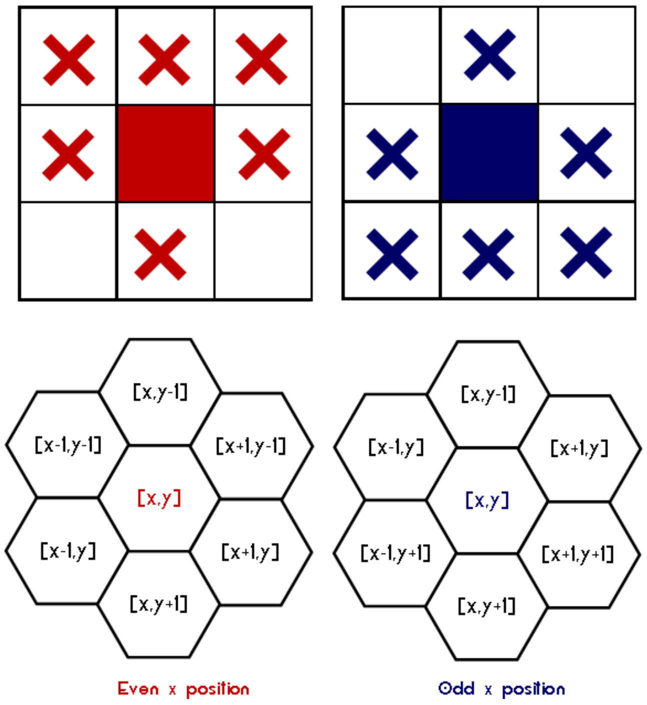


Figure 4.6 Neighbors of an hexagon according to its X position both in the matrix as graphically in the grid.

Depending if the horizontal position of the hexagon is odd or even, it will have its 6 neighbors. It can be seen that an hexagon (in the squared grid) is surrounded by 8 hexagons. In both cases of even and odd positions, they share that they have neighbors at the left, right, up and down. If its position is even, it will have as two extra neighbors the upper corner hexagons. In the odd case, it will have the lower corner.

For a better understanding of how those neighbors are chosen, they can be seen in the figure the hexagons placed visually and their corresponding coordinates in x and y according to the central tile.

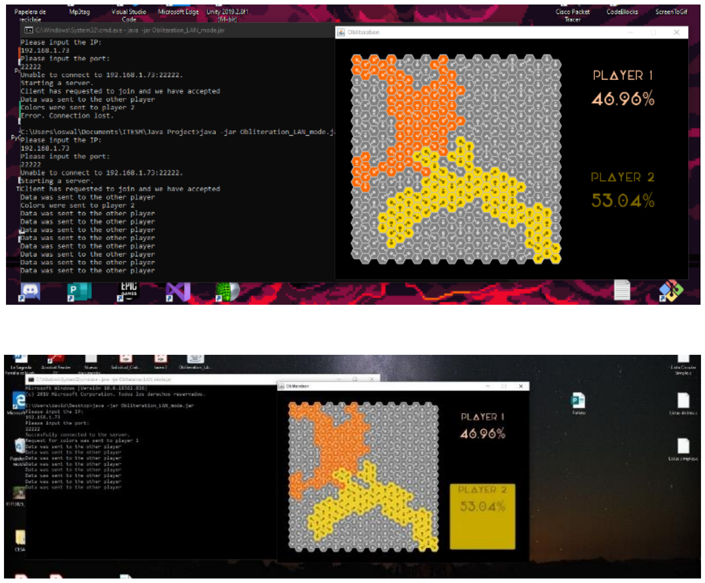
If one of these rotations is pointing outside the grid, instead coordinates, the value is null, so the conversions' algorithm avoids using them.

5. RESULTS

This first execution example shows the game running on two different computers. Both players are connected to the same network.

Each player can only click their corresponding hexagons. For example, player 1 can't rotate player 2's hexagons, it only can be done on the screen of the player 2.

Also, a player cannot rotate their own hexagons if it's not their turn. In the screen, the turn of the player is visible when a colored rectangle appears behind the score and name of the player. In the figure, for example, it's the turn of player 2, but only that player can see that it's their turn. The player 1, on the other side, doesn't see any rectangle.



C:\Windows\System32\cmd.exe - java -jar Obliteration_LAN_mode.jar
Obliteration
PLAYER 1
46.96%
PLAYER 2
53.04%

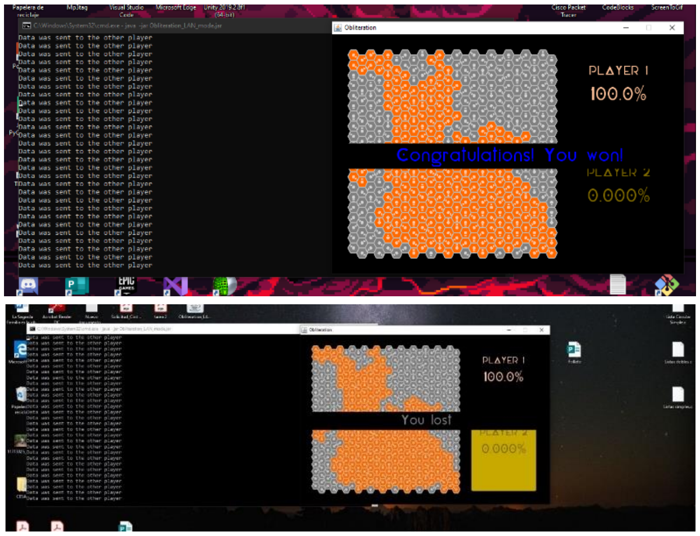
```
Please input the IP:
192.168.1.73
Please input the port:
22222
Unable to connect to 192.168.1.73:22222.
Starting a server.
Client has requested to join and we have accepted
Data was sent to the other player
Colors were sent to player 2
Error. Connection lost.

C:\Users\oswal\Documents\ITESM\Java Project>java -jar Obliteration_LAN_mode.j
Please input the IP:
192.168.1.73
Please input the port:
22222
Unable to connect to 192.168.1.73:22222.
Starting a server.
Client has requested to join and we have accepted
Data was sent to the other player
Colors were sent to player 2
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
```

```
Microsoft Windows [Versión 10.0.18362.836]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\david\Desktop>java -jar Obliteration_LAN_mode.jar
Please input the IP:
192.168.1.73
Please input the port:
22222
Succesfully connected to the server.
Request for colors was sent to player 1
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
```

Figure 5.1 Game execution in two different computers. It's the turn of player 2.



C:\Windows\System32\cmd.exe - java -jar Obliteration_LAN_mode.jar
Obliteration
PLAYER 1
100.0%
Congratulations! You won!
PLAYER 2
0.000%
You lost

```
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
```

```
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
Data was sent to the other player
```

Figure 5.2 Game execution in two different computers. Although it's the turn of player 2, all their hexagons were converted, so the game ended, declaring the player 1 as the winner (up) and player 2 as the loser (down).

The following figure shows the game running twice on the same computer.

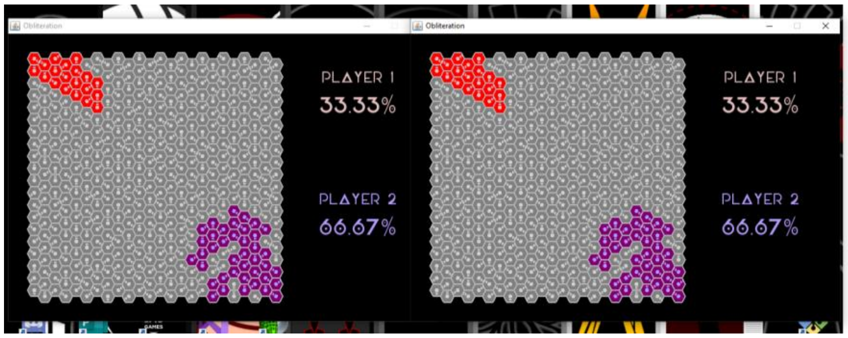


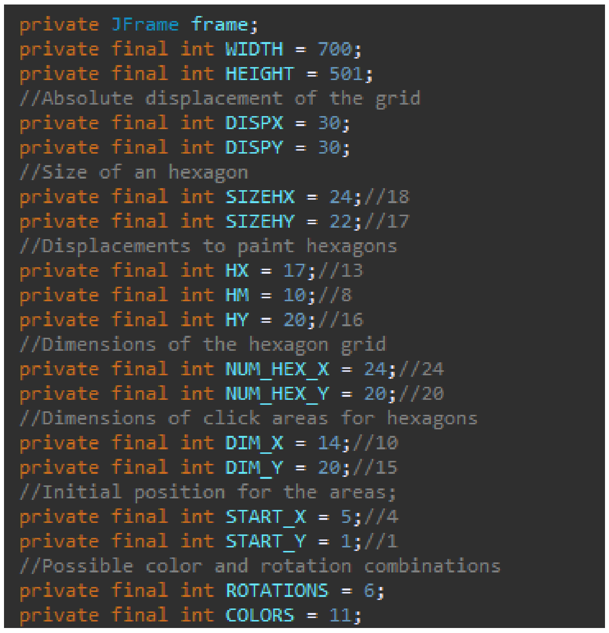
Figure 5.2 Game running on a single computer. The game can be executed by just double clicking the file because this version doesn't need a console to work.

As explained before, the turn of the player is visible with a colored rectangle, but in this screen, none of them is showing any. This is because, according to the flow diagram, the turn of the player is given once the conversions are made. In this example, player 2 took their turn and purple hexagons haven't finished converting all connected tiles. Once they are done, the game reads which player sent the String. Player 1 read that the String came from player 2 through a socket, so it interprets it's their turn. Player 2 read that the String came directly from one of its clicked hexagons, so it waits until player 1 makes their movement.

6. CONCLUSIONS

Although the rules of this game are as simple as Domino or Tic-Tac-Toe, it's evident the difficult to implement it physically due the big amount of movements needed on each turn. That's the reason why this programmed version is more efficient and easier to play (and maybe the only viable option).

Some minor improvements can be that players may choose the size of the grid, their initial colors and even the size of the hexagons. That's easy to do since in the game are some final variables that control those settings.



```
private JFrame frame;
private final int WIDTH = 700;
private final int HEIGHT = 501;
//Absolute displacement of the grid
private final int DISPX = 30;
private final int DISPY = 30;
//Size of an hexagon
private final int SIZEHX = 24;//18
private final int SIZEHY = 22;//17
//Displacements to paint hexagons
private final int HX = 17;//13
private final int HM = 10;//8
private final int HY = 20;//16
//Dimensions of the hexagon grid
private final int NUM_HEX_X = 24;//24
private final int NUM_HEX_Y = 20;//20
//Dimensions of click areas for hexagons
private final int DIM_X = 14;//10
private final int DIM_Y = 20;//15
//Initial position for the areas;
private final int START_X = 5;//4
private final int START_Y = 1;//1
//Possible color and rotation combinations
private final int ROTATIONS = 6;
private final int COLORS = 11;
```

Figure 6.1 Final variables in class Obliteration (JPanel).

In fact, in the folder 'res' on the repository, there are two files called 'Hexagons.png' and 'HexagonsM.png' where 'M' is for 'medium'. The reason of this was that the first of the files had very small sprites and it was difficult to click the desired. The second image has bigger hexagons that just needed to have different values on some of those final variables to be easily shown in the grid.

Finally, there are some bigger improvements that could be applied in the future like allowing to enter more players or use brokers so the game can be played on different networks.

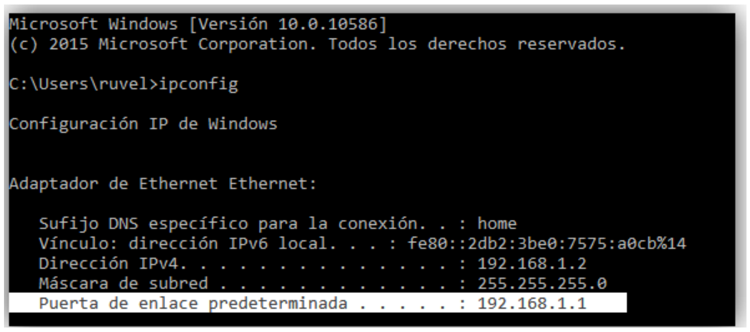
7. LAUNCH INSTRUCTIONS

The project is licensed under the **GNU General Public License** that can be consulted on https://www.gnu.org/licenses/gpl-3.0.txt or in the repository of this project.

There are two versions of the game available in the repository https://github.com/Oswaldo1002009/Obliteration_JAVA. The first is a file called 'Obliteration_local_mode.jar' and can be opened just by double-clicking the file twice on the same computer.

The file 'Obliteration_LAN_mode.jar' can be played using two different computers connected to the same network. To play it, follow the next steps:

1. In one of the computers, open the command prompt, type the command 'ipconfig' and copy the IP.
2. Open the command prompt on the second computer.
3. On both computers, go to the folder of the game in the console and type the command 'java -jar Obliteration_LAN_mode.jar'.
4. Type the IP copied from the first computer.
5. Type a port number between 1 and 65535. Both computers must have the same number. The recommendation is **22222**.
   - If you get a message like: "Error: 192.168.1.1:22222 doesn't exist or is already in use." you could write wrong the IP or port. There can be cases where there are already two players in the game, or the port is in use by the computer. In those cases, try again with a different port number.



```
Microsoft Windows [Versión 10.0.10586]
(c) 2015 Microsoft Corporation. Todos los derechos reservados.

C:\Users\ruvel>ipconfig

Configuración IP de Windows


Adaptador de Ethernet Ethernet:

   Sufijo DNS específico para la conexión. . : home
   Vínculo: dirección IPv6 local. . . : fe80::2db2:3be0:7575:a0cb%14
   Dirección IPv4. . . . . . . . . . . . . . : 192.168.1.2
   Máscara de subred . . . . . . . . . . . . : 255.255.255.0
   Puerta de enlace predeterminada . . . . . : 192.168.1.1
```

Figure 7.1 Example of an IP

REFERENCES

Emanuele Papele. (2016). *Elianto - Free Font on Behance.* Retrieved from: https://www.behance.net/gallery/33808280/Elianto-Free-Font

Lomaz. (2018). *Corruption.* Retrieved from: https://www.newgrounds.com/portal/view/714303

Lomaz. (2018). *Corruption II.* Retrieved from: https://www.newgrounds.com/portal/view/714635

Oracle. (n.d.). *Lesson: All about sockets.* Retrieved from: https://docs.oracle.com/javase/tutorial/networking/sockets/index.html

Oracle. (n.d.). *How to use Panels.* Retrieved from: https://docs.oracle.com/javase/tutorial/uiswing/components/panel.html

Oracle. (n.d.). *Lesson: Concurrency.* Retrieved from: https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html