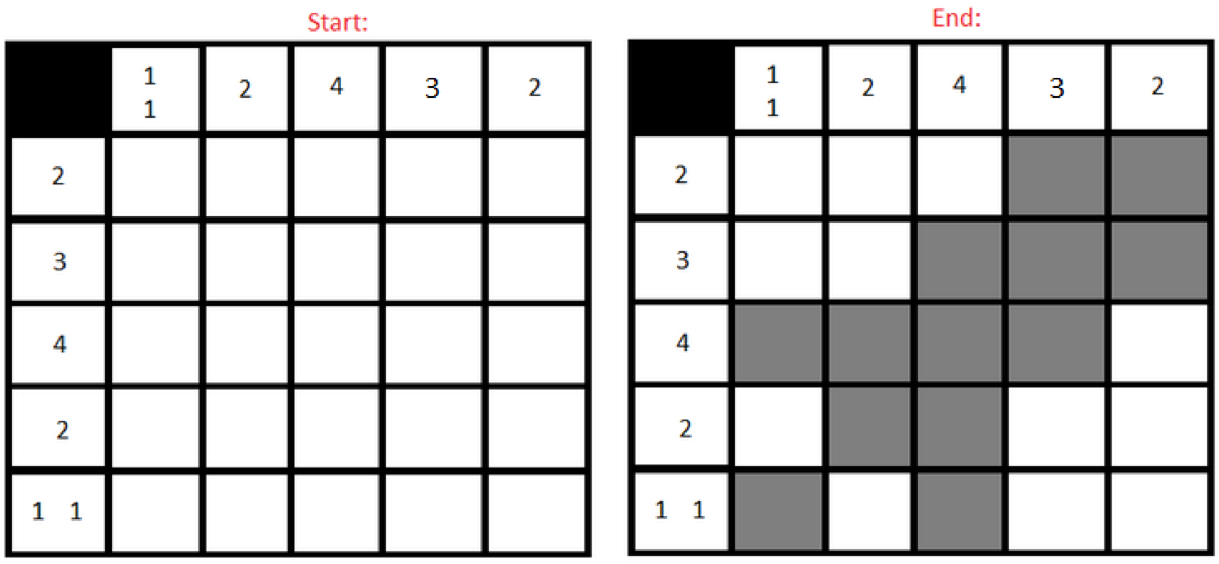
Picross

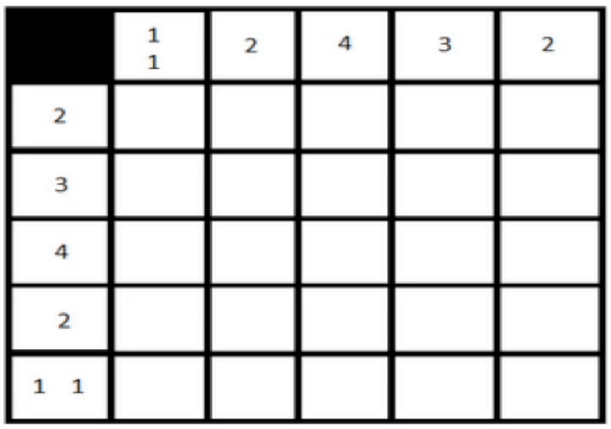
Andre Lenoir, David Koss, Jessica Llewellyn, and Samuel Jochum

What is Picross:

Short for "Picture Crossword", Picross is a logic puzzle game in which you are tasked with constructing a pixel image based on values at the side of a m x n grid that was created in 1987 independently by both Non Yoshida, a japanese graphics editor, and Tetsuya Nishio, a japanese puzzle creator (Source) . There are many ways in which this game can be played but most often it is done in a binary of black and white tiles and will often have a freeplay or an image mode. The following image is a sample picross board and completion:



While quite simple at surface level, it is a game that is exceedingly hard to master. There exists many techniques that need to be employed to solve a puzzle. The following is the first steps to complete the puzzle above:



As the board is 5x5, the player knows that both 4's must take up the middle 3 of their respective rows and columns. From this the player is able to remove other positions that cannot be reached by 2's on the board, mark them red, and then finally conclude that on the top left and bottom right there are not two open squares for them to be able to place 2 gray tiles and hence they are marked red as well. This is one of the simplest strategies needed to complete a game and there exists many more complex strategies that become more prevalent for larger boards.

It is clear to see why picross can be so fun with different size boards for skill expression and the strategy needed to solve even the simplest board. It is no wonder then that since its creation in the late eighties that it has been adapted into many different video games with many different intellectual properties serving as the basis for said games such as Mario, Zelda, Sanrio, and Pokemon.

After creating a list of possible apps including but not limited to a calendar, a wordle like game ie something akin to nerdle, a fitness app, and a bus scheduler one of the members offered up the idea of Picross. After demoing a playthrough and explaining the rules of the game, we decided that this would be our project.

Goals:

We had 4 main goals when we initially started this project:

1. Game Completion
2. Free Play Mode
3. Campaign Mode
4. Board persistence

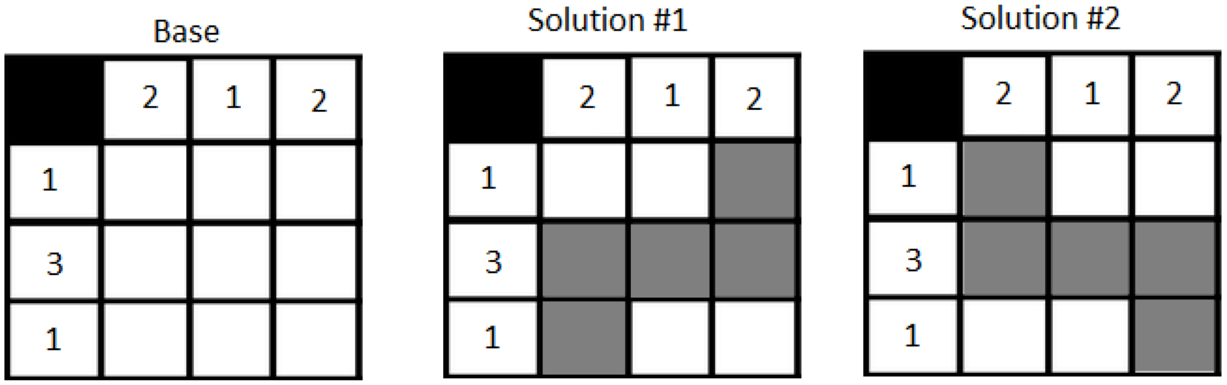
Additionally we had the following 3 stretch goals:

1. Leadership Board
2. User customizable boards
3. Implement colored blocks

In the following section we will describe these goals in further detail and how we went about implementing them.

Game Completion:

Although it may seem simple, game completion was obviously one of the most important issues to square away first. It needed to be fast computationally as we would need to check it continuously throughout the game and arguably more importantly allow for the possibility of multiple solutions to a single board. One of the key features in picross is the fact that there may exist multiple solutions to a board which makes it very important that all solutions are recognized, especially in a free play mode where boards with multiple solutions become more prevalent. The following image is a simple example of a 3x3 board with two solutions as well as it's solutions shown:



To this end, our first implementation of board completion was simply to check each cell against a known completed board after a cell was updated. While this was computationally efficient being bounded above by $O(m * n)$ but being much lower in the average case, it did lead to an immediate and glaring problem in that we were not allowing for the possibility of multiple solutions.

We now had to decide on a tradeoff. Since we did not know whether a board could have multiple solutions by looking at the board without supplemental algorithms that would take much longer than $O(m * n)$, we must either accept that a user may have a completed puzzle but our game does not recognize it or two rework our implementation for multiple solutions at the cost of the algorithm always running in $O(m * n)$. Of the two we chose the latter.

Our group member Andre recognized that using functions that we created within the first milestone "getRowString" and "getColString," discussed later in more detail, we need simply compare the known completed board's set of row and column strings to the player board's set of row and column strings and see if they matched. As stated previously, with how this is implemented we were now going through the entire matrix, in fact twice, each time an update was done to the board but it would solve the problem of multiple solutions not being recognized.

One might argue that it may have been simpler to add a button to the game, say within a tab view, that would do this instead of continually doing it for each board update. While this is a fair criticism, we'd rather have it such that the game completion is handled in the background without the player being aware of it going on and ultimately for the small board sizes which we have implemented the computational expense is overall still low enough for it to not matter.

Free Play Mode:

Our second goal was to make a random solvable board for players to play on. This ultimately was not difficult only needing for us to create a function that takes parameters m and n both positive integers, creates a board of size m x n, and finally populates it with random values to create a new board. The row and column strings of this board would then need to be visible to the user.

Campaign Mode:

Campaign mode is a mode that is very common within Picross games. The idea of it is for the player to go through a sequence of predesigned templates that are associated with some sort of pixel art. Our initial idea was to implement this with a set of premade boards that could be chosen from a subview within our app. How we had planned to implement this was to use a file manager and to make boards encodable, encode in a small set of boards, and to implement a tabview through which would have lead to a navigation view in which you could find these campaign boards; however, sadly classes picked up for group members and resulted in a lack of time, we did not meet this goal. To supplement it however we instead focused on game board persistence and our stretch goal of a leaderboard for players to compete against themselves for their best time in freeplay mode.

Board Persistence:

We wished to have board persistence between opening and closing of the app so that players could resume their free play session. To do this, we used user defaults to save the last used board instance by the player. This was one of the last of our major goals that we implemented.

Stretch Goals:

Of our stretch goals we were only able to make one in a leaderboard again due to time constraints of group members. The other two goals were user created boards and to implement a non-binary mode where players would have to place different colored tiles to progress.

The first of these relied on campaign boards being implemented which again was cut due to time constraints on the group. The second of these, non-binary mode, however was a lofty goal and would likely only be used in non-freeplay modes. This is due to how our implementation of establishing board completion was implemented; we would likely need there to be only one board completion state for it to work effectively. Further we would need to retool our implementation of establishing rows and column strings and hence our implementation of checking board completion would also need to be retooled as it is reliant on the former. Put simply, it would require a lot of changes to our codebase and hence it went unprioritized.

Leftovers of this idea can still be seen within enums ColorTile and Mode of game.swift from initial development.

Demo:

https://www.youtube.com/watch?v=fk1oisZBsuw

Timeline:

In the following section we would like to discuss the development timeline of our app, each milestone's goals and pertaining design documents will be featured.

Milestone 1: Implementing Game Logic

We began with a simple design document created in paint discussing the critical needs of our game. Those being creating and updating the board, and game completion. Below you can see the full document.

$$\text{board} = \begin{bmatrix} a_{1,1} \cdots & a_{1,n} \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ a_{m,1} \cdots & a_{m,n} \end{bmatrix}$$

Class / Struct [needs to be mutable]
board an m x n integer matrix
has property of being comparable [player board vs true board]
integer for our strectch goal of colorable

Board Creation Functions:
createRandomBoard(int mode, int m, int n)
- creates m x n board for freeplay, mode var for stretch goal of colorable, assumed 2 for black and white.

createCampaignBoard(? ? ?)
- creates board based on preset board. Need to create database for this. don't know what inputs will be.
- possible strech goal method for using user images for boards here too

Player Functions:
updateBoard(int x, int y)
- updates board to be darkened or not based on position of point.

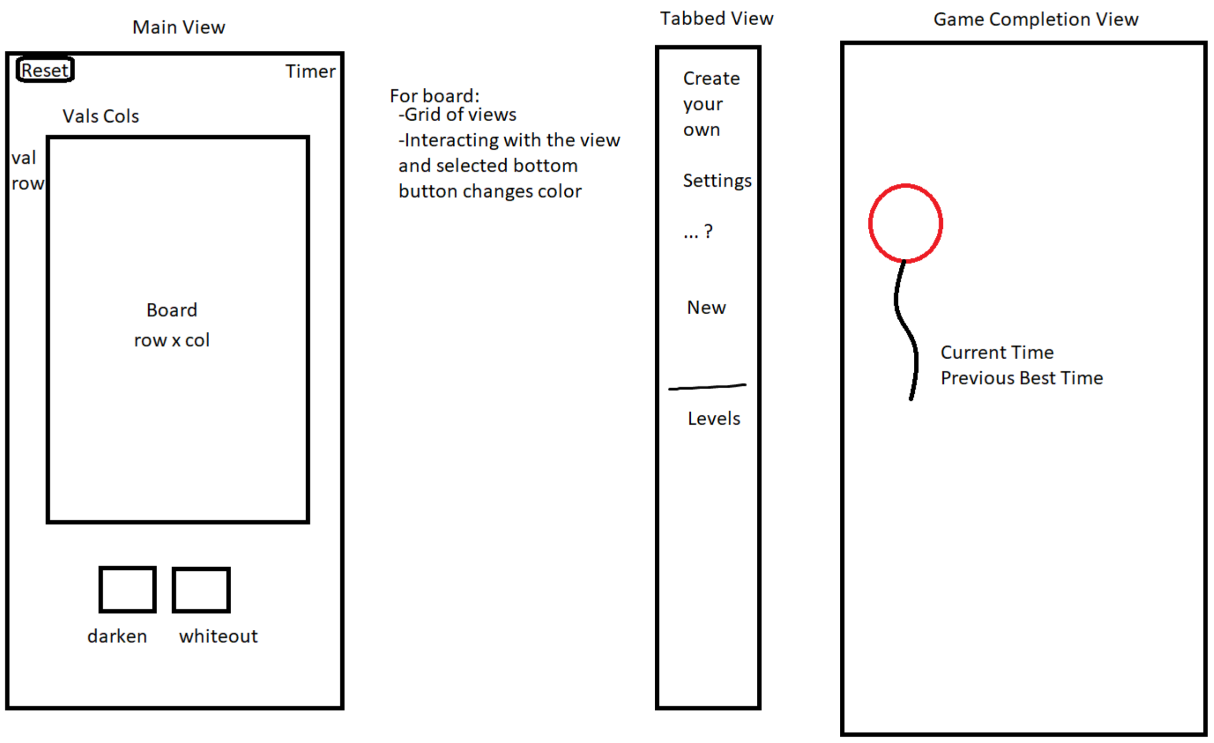
updateRow(int x, int y1, int y2)
- updates full row to be darkened or not based on row x and points in row y1, y2. Similar method with updateColumn(...)

compare(Board b1, Board b2)
- compares player and final boards element wise. Plenty of optimizations possible but will be needed to be called for each update to the board, thus should be reasonably quick.

To the end of implementation for this, we completed createRandomBoard, updateBoard, and worked on board completion although we'd later refine it in milestone 3 to work for multiple solutions. We chose not to include updateRow and instead to have each cell be updated individually. Additionally we added "getColString" and "getRowString" functions which are very self descriptive in title but to the point they will create a string that tells the user how many filled blocks are within a row or column according to the rules of picross.

Milestone 2: Implementing UI

- For this milestone the group got together and decided what we want the final look of the app to be. In paint we designed the following:

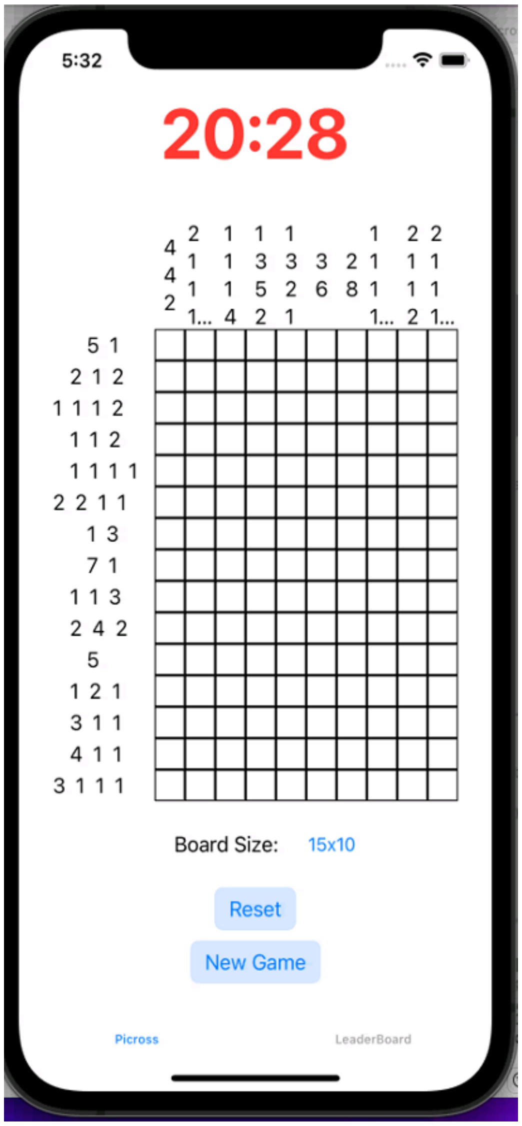


- Once we made this design, we began to code the features.
    - The rest button was a Button View that when pressed cleared the tiles on the board to white but kept the same final board.
    - A timer that started when a new game began and counted up till the game was completed.
    - A board with m by n tiles and corresponding numbers on the top and left side of the board that corresponded to how many tiles should be filled in. Originally we found the numbering part to be a little tricky to correspond with the spacing and alignment of the board however in the end we were able to get everything to line up even when the size of the board was changed.
    - Dark/light buttons that were 2 Button Views that changed the color of the tile being pressed, i.e. if the dark button was pressed then a certain tile was pressed, the tile would then change to black.

  - We then thought of designing a tab view that allowed the user to do a variety of actions, in the end the tab view included a board tab and a leadership board tab.
- In milestone 2, we were originally hoping to implement persistence to save the board if the app is closed, this was implemented in milestone 3.
- In the end our final layout did not look the exact same as our original, but the different features were all still available.

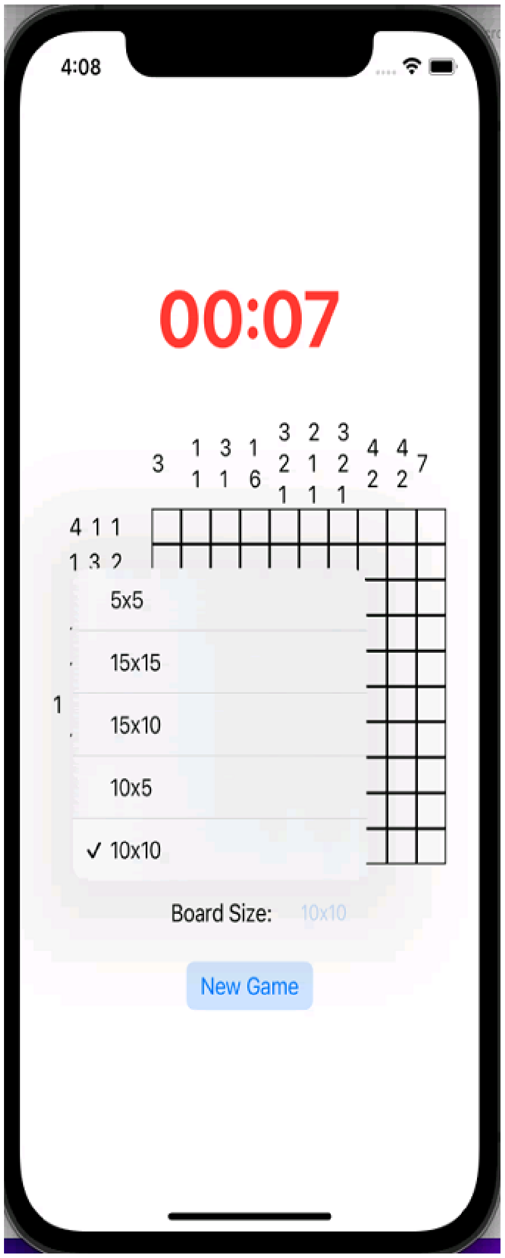
Milestone 3: Polish off Gameboard, Introduce Premade Boards and Campaigns

- The game board was able to be polished off, which makes it a lot nicer than it did in milestone 2.



- We were also able to create premade boards that are randomly generated. Upon hitting the new game button, it will generate a new random board.

- We were able to implement a picker that allows us to choose different board sizes that we want, such as 5x5, 10x10, or even 15x10.



- Another implementation that we did was implementing “saving the state of the board” which was a milestone 2 objective but we ended up carrying it over to milestone 3.

Future Directions:

There were features on here that we weren’t able to implement. Going back to our stretch goals, one of our objectives was to implement colored blocks, which we weren’t able to do. This would be one of the implementations we would start off with if we were to continue on this project. The colored blocks allow for a smoother visual experience for the user and will help with clearly identifying the boundaries of the square regions once filled. Another nice feature would be adding a menu which would allow you to select different modes. Right now when the app is opened it brings you right into a randomly generated board with the timer starting. So polishing up this app would begin with a nice menu scene. We also would like to add different modes and gametypes. This would diversify the app a bit and allow for better user progression depending on the gametype they choose. The leaderboard we added is a very nice start, but expanding on that would create a more professional feel. We could add a database and create global rankings among players. This would create a competitive feel for the users which would give more reason to play. Casual modes of course would be there too, they would just be expanded from what we already have. There’s just so much potential with this app and so many different directions we could take it. Overall, we feel this app is a very good start to something that could be a lot more given more work and time added to the development of the app. But we’re very happy with what we were able to accomplish with the limited time given.