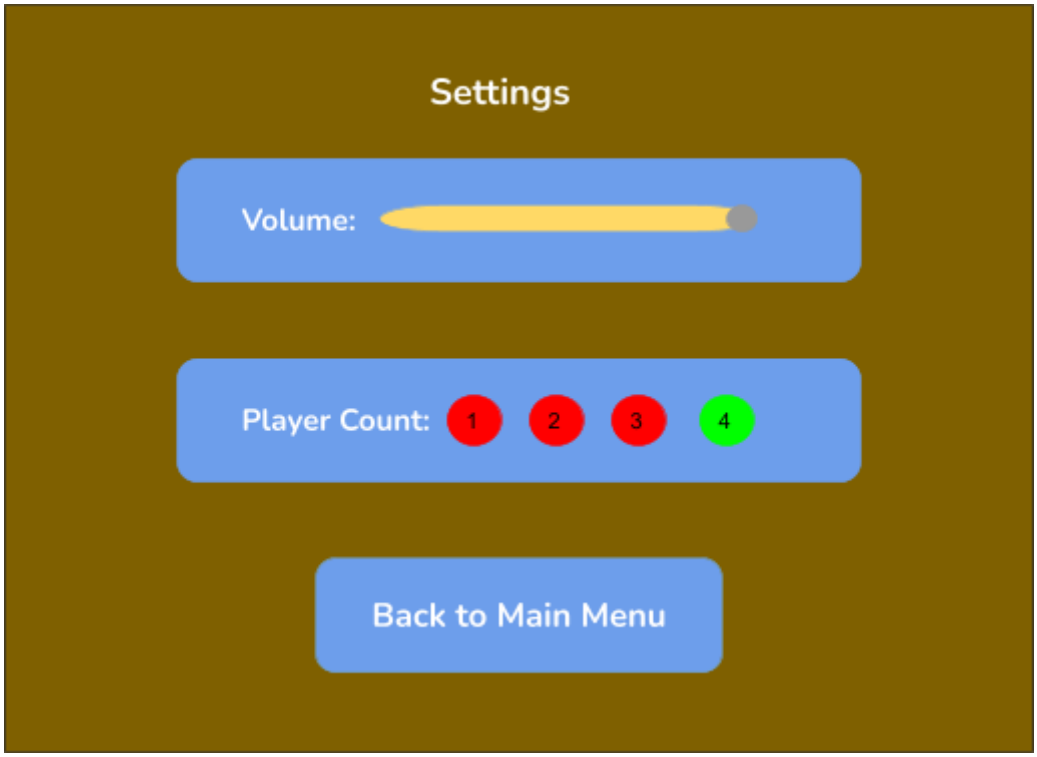
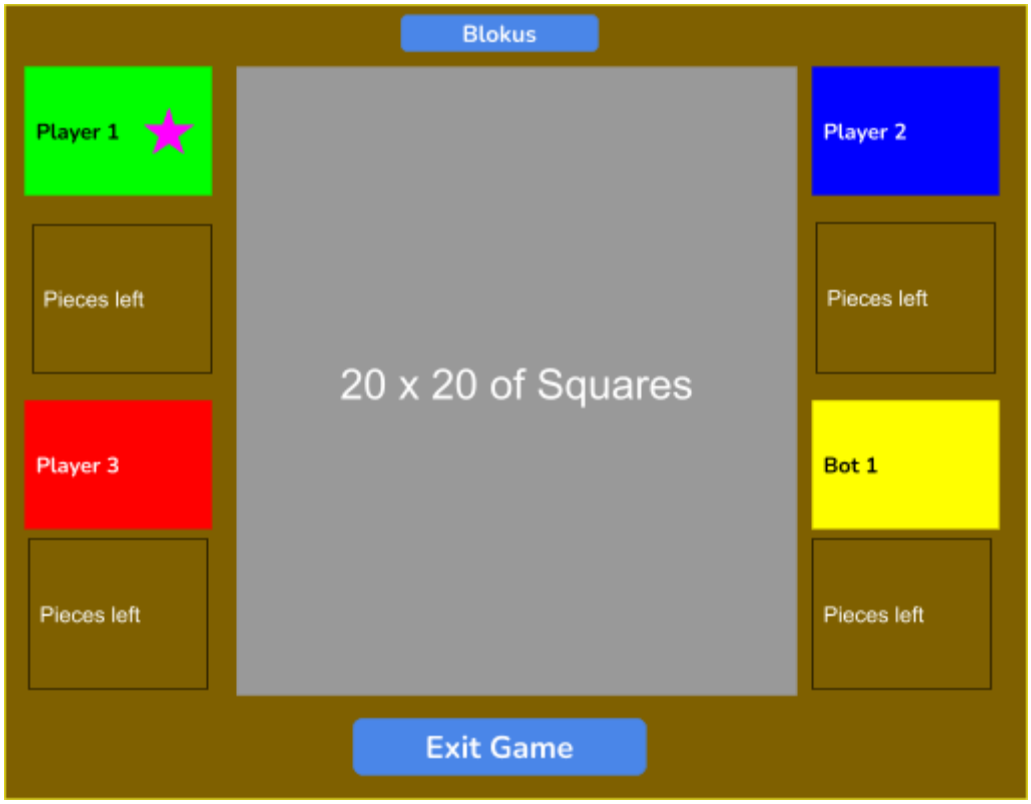
Prototype Documentation

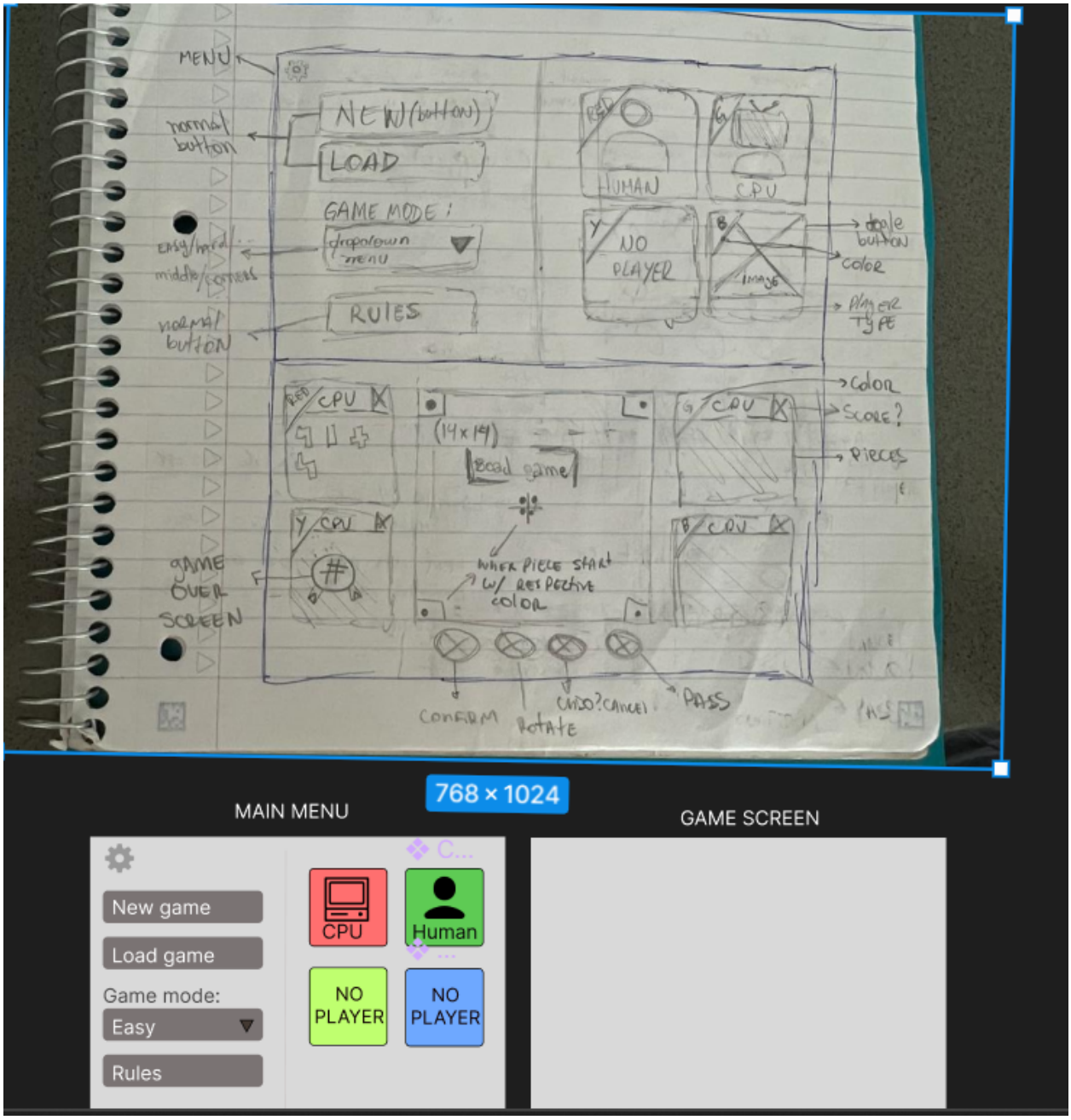
Below are the draft prototypes made by each of the group member in order to brainstorm and decide on how the final prototype would look like.

Justin:

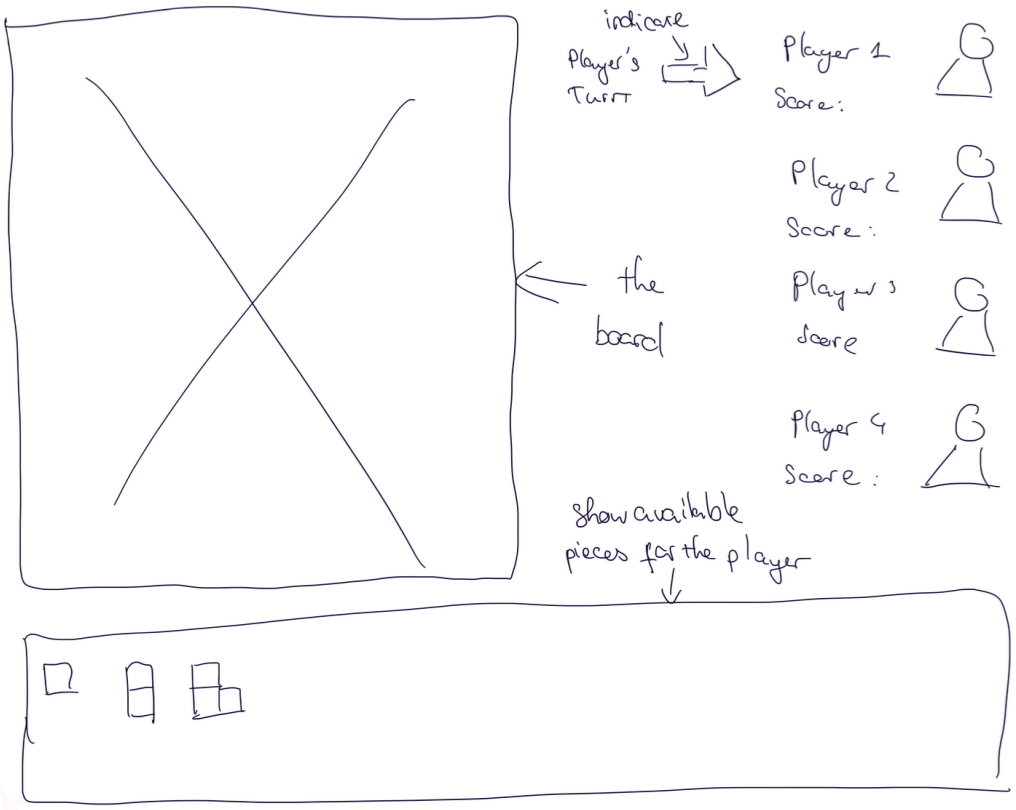



Blokus
Player 1
Player 2
Pieces left
Pieces left
20 x 20 of Squares
Player 3
Bot 1
Pieces left
Pieces left
Exit Game

Joao:


MENU
normal button
NEW(button)
LOAD
GAME MODE:
dropdown menu
easy/hard
middle/corners
normal button
RULES
HUMAN
CPU
NO PLAYER
toggle button
color
Player Type
CPU
(14x14)
load game
color
score?
pieces
GAME OVER SCREEN
WHEN PIECE START W/ RESPECTIVE COLOR
CONFIRM
ROTATE
PASS
768 × 1024
MAIN MENU
GAME SCREEN
New game
Load game
Game mode:
Easy
Rules
CPU
Human
NO PLAYER
NO PLAYER

Van:



BLOKUS

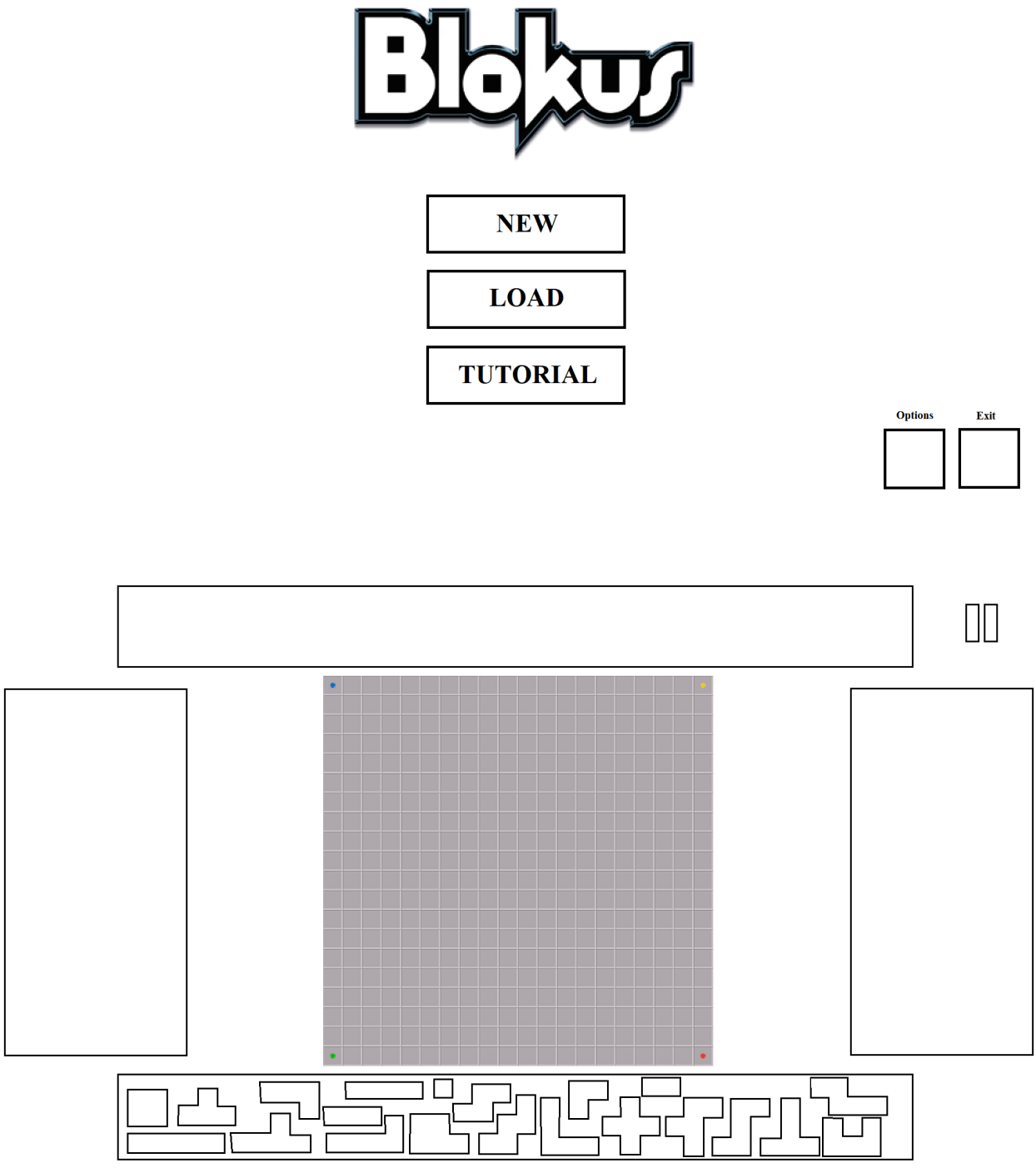
New Game

Load Game

How to Play

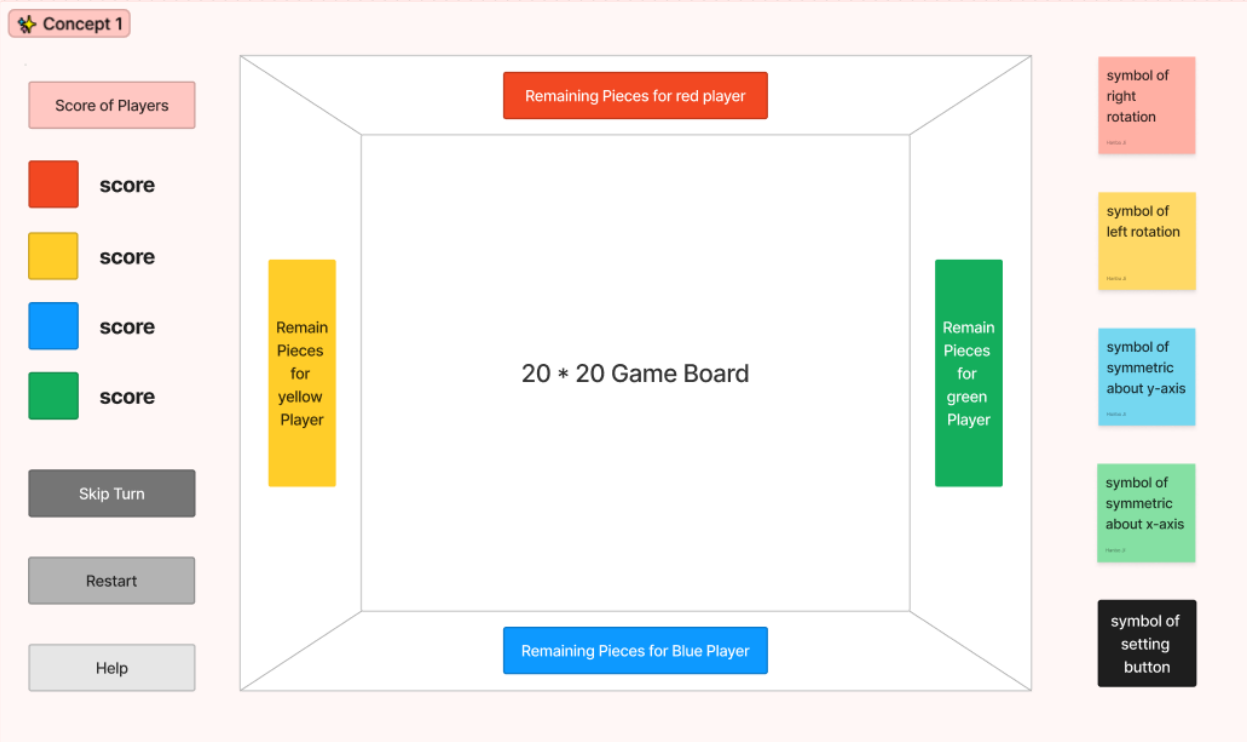
Settings

Anton:

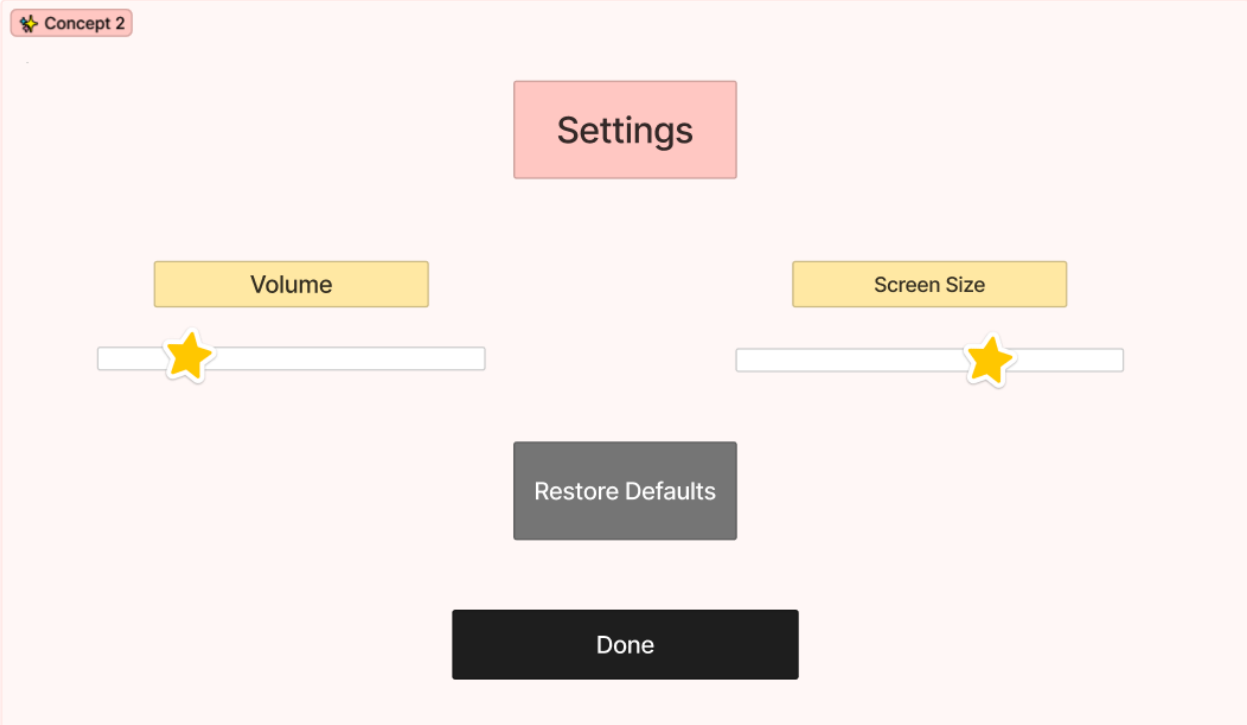


Hanbo Ji:

1) Game Playing Interface



2) Setting Interface



These are the main comments for each prototype.
Main menu:

- Joao:
    - Advantages:
        - Communicate state clearly
        - Gives the user control to set how many players and which one of those players are human
        - Consistent font and colours
    - Disadvantages:
        - Clustered with information
        - No achievements button
- Van:
    - Advantages:
        - Communicate state clearly
        - Gives the user control
        - Consistent font and colours
        - Not overwhelming with information
    - Disadvantages:
        - No exit game button
        - No achievements button
- Anton:
    - Advantages:
        - Communicate state clearly
        - Gives the user control
        - Consistent font and colours
        - Not overwhelming with information
        - Centers attention on new game or tutorial for new players by emphasizing those buttons.
    - Disadvantages:
        - No achievements button
        - No colour

Settings:

- Justin
    - Advantages:
        - Communicate state clearly
        - Gives the user control
        - Consistent font and colours
    - Disadvantages
        - Missing some settings
        - The back to main menu button could be in a different colour box or font colour
- Hanbo
    - Advantages:
        - Communicate state clearly
        - Gives the user control
        - Consistent font and colours

    - Disadvantages
        - Instead of Done, go back to Main Menu is clearer

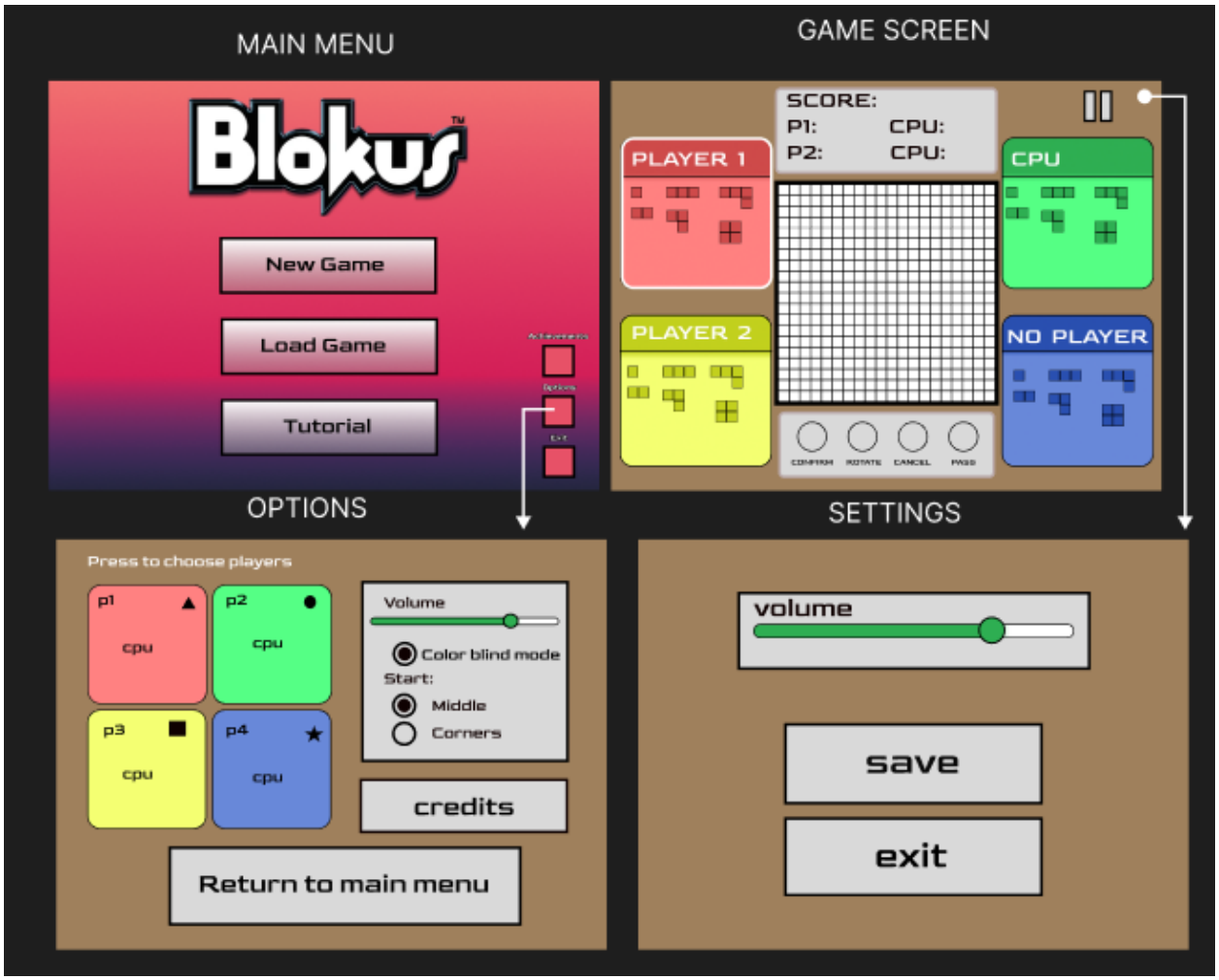
Board Game:

- Justin
    - Advantages:
        - Communicate state clearly
        - Shows all pieces of each player
        - Consistent font, colours and box sizes
    - Disadvantages:
        - Doesn't allow the player to rotate or place a piece
        - Could be clustered with information and boxes
        - Colour Scheme could be better
- Joao
    - Advantages:
        - Communicate state clearly
        - Gives the user control of how they want to place piece
        - Consistent font and boxes
    - Disadvantages:
        - Could be clustered with information and boxes
- Van:
    - Advantages:
        - Communicate state clearly
        - Consistent font and box sizes
        - Interface is not that clustered
    - Disadvantages:
        - Cannot see other players pieces when its your turn
        - Doesn't allow the player to rotate or place a piece
- Anton:
    - Advantages:
        - Communicate state clearly
        - Consistent font and box sizes
        - Interface is not that clustered
        - Pause button
    - Disadvantages:
        - Cannot see other players pieces when its your turn
        - Doesn't allow the player to rotate or place a piece
        - Doesn't show all players in the game
- Hanbo
    - Advantages:
        - Communicate state clearly
        - Consistent font and box sizes
        - Use proper color for interactables / non-interactables
        - Has all the features of the game

- Disadvantages:
  - Somewhat clustered and can be overwhelming to the user

Final Design:

For the final design, we incorporated the advantages of each prototype so that the interfaces are clear and easy for the user to understand. Starting with the main menu, we took ideas from Anton and Van's prototypes because they had an advantage in clarity and attention. For example, their prototypes had buttons to start the game in the center while options and paraphernalia like achievements are in the corner. For the settings interface, we can add additional settings such as a toggle for color blind mode. Also for focus/attention, the back to main menu button could be in a different box or have different font colour. Lastly for the gameplay interface, we are going with a design similar to Justin and Joao's where the players can see everyone's pieces at all times. We also want to have buttons for placing, rotating and changing the pieces.



After bringing the prototype from paper to code. We develop a playable prototype of how our final game will look like. Some design patterns were used in this phase of the project and below are them and how they incorporate our programming.

Design Pattern:

Command:

1. Implementation

These functions: on_exit_clicked(), next_player_clicked(), and confirm_placement() in gameInterface.py are similar to Command objects in Command design pattern. They encapsulate “Confirm” and “Exit” actions and provide Python GUI to execute these actions. In gameInterface class, QpushButton widgets have functionality as invokers. They carry commands and execute the command method when the user clicks related buttons. Besides, the gameInterface class can be seen as the receiver, to hold game’s stated and UI elements the commands manipulate.

2. Reason

Firstly, this pattern separates QPushButton widgets from commands. This setup separate initialization of an operation from invoker(QPushButton widgets) from the entity that how to carry it out. Leading to more flexibility for the code, so that I can change the commands tied to button clicks without tweaking code handling the button.

Secondly, Command design pattern is easy to add extra new commands. I can simply achieve this by creating new functions and linking it to a button click.

3. Improvement

Increasing Extensibility: If you want to add new functionalities or commands, you can do so by defining new functions and linking them with appropriate invokers (buttons). This pattern allows me to easily make extensions of the codebase in further development without having to make major changes in the existing structure.

Reduction in Code Complexity: By isolating commands in separate functions, it reduces the complexity of the codebase. Each function is responsible for its own behavior, meaning that bugs are easier to track down, and the codebase as a whole is easier to test and debug.

Builder:

1.Implementation:

The builder design pattern is implemented in the piece, board, playerPanel, turn and player classes because they work together to build a game interface that the user can interact with. These classes can be found in the board.py, pieces.py, players.py and gameInterface.py files.

2.Reason:

We chose to use a builder design pattern because it makes the code in our gameInterface class easier to understand and less redundant. Instead of doing everything in the gameInterface class, we made separate classes for each component of the interface. This way we are each able to work on a different part of the interface and combine the components at the end.

3.Improvement:

The advantage of using the builder design pattern is that it allows us to create a gameInterface object step by step. A builder design pattern also allows us to isolate individual components of the game interface that we want to modify. The advantage of this is that we do not have to change the gameInterface class and only need to modify the class we want to change. For example, to make a change to the colour of the board we only need to change the board class code.

Flyweight:

1.Implementation:

The flyweight pattern is implemented in the Piece class by caching the QPixmap objects in the pixmap_cache dictionary. The dictionary serves as a shared repository of pixmap data, allowing instances of the Piece class to reuse existing pixmaps instead of creating new ones for each instance.

2.Reason:

To optimize memory usage by sharing common pixmap data among multiple game piece instances. Since the pixmap data for game pieces can be quite large and there can be many instances of the same pixmap, sharing the data helps reduce memory consumption.

3.Improvements:

1. Reduce memory: By sharing the pixmap data through the flyweight pattern, the memory usage is optimized as multiple instances of the same pixmap can reuse the shared data instead of duplicating it.
2. Faster creation: Since the flyweight pattern eliminates the need to create new pixmap objects for each instance, the object creation process becomes faster, resulting in improved performance during initialization and gameplay.

Overall, the implementation of the flyweight pattern in the Piece class provides memory and performance benefits by reusing shared Pixmap data, resulting in a more efficient and optimized game interface.