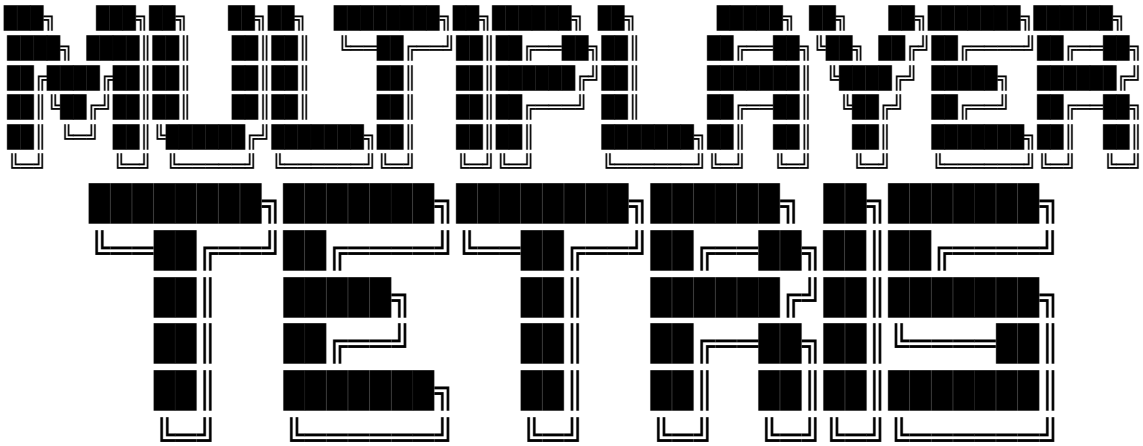


Amelia Cook, Alex Williams Ferreira, Trevor Sullivan

Table of Contents

Project Proposal

Team Name

amexor

Group Members

Amelia Cook
Trevor Sullivan
Alex Williams Ferreira
Elliot Bonner

Project Description

Concurrent Tetris!
Our project will be a multiplayer tetris game hosted on the halligan server. The server will allow people to either join public lobbies with other random players or create a private lobby that friends can access using its name. There will be multiple game modes available to play. The three game modes will be **Head-to-Head**, **CHAOS MODE**, and **Tetris21** (directly inspired by Tetris99). All games will be based on point systems to determine a winner, but **Tetris21** and **Head-to-Head** also have knockout properties. **Head-to-Head** mode has one large board cut up by n users. Traditional tetris is played, but rows must be cleared across all players. The player who clears the row gets the points for it. **CHAOS MODE** will have all players playing on one board sized by the number of players. All players can place blocks at any point, those who clear rows get the points. **Tetris21** will be a simplified version of Tetris99. Each player will have their own board and when players clear rows, rows get sent to other players' game boards as punishment. This game is played battle royale style, and the final person standing wins. There will be a leaderboard that keeps track of top scores across the games!

MVP Description

Our minimum deliverable would be a server that supports multiple players in multiple rooms. This deliverable will have a GUI to represent the game play, likely in terminal. Our default game mode, Head-to-Head, will involve multiple processes playing on the same tetris board. In this mode, the board will be wider to accommodate the additional players and each player will be given a section that it may place blocks in. Players will be able to place blocks concurrently, but the points will be given to the player who places the last block to clear a row. Should any player lose, their section will become full of blocks and the game will continue for the other players.

If we have enough time, we would like to implement additional game modes. CHAOS MODE will be similar to Head-to-Head, but the players would not be restricted to sections of the board. This would create some interesting concurrency challenges to avoid blocks being placed on top of each other, for example. Tetris21, inspired by Tetris99, would give each player their own board but allow them to see

other players' boards. When a player finishes a row, they can target other players and give them additional rows as an additional challenge.

An additional function we would like to implement is a global leaderboard for each game mode across all players and rooms.

First Steps

Our first step is to design message protocols. After that, we plan to set up the server and begin developing the GUI. We hope to build a normal game of tetris on the foundations of our concurrent multiplayer game to make sure we have things working.

Biggest Problem or Questions

We foresee two main problems: designing/implementing a GUI, and keeping a server running. We want to have a server running continuously for a smooth user experience. We don't think this will be difficult to solve–we just don't know how the logistics would work. Our next (and biggest problem) is designing a GUI. We want to make it terminal-based, but we don't have much experience designing GUIs. It's going to be difficult to figure out how each player interacts with the GUI, and what the GUI displays for the other players (e.g. should we update the GUI every time another player rotates their block, or should we only update it when they successfully place a block?). Since our project is multiplayer, it is critical that the GUI updates as quickly as possible since any delays will hinder the players' experience.

Initial Design

Design Decision Explanation

One of the design decisions we made for this program was regarding the server of the program and how it would interact with the game. Overall our question was what the source of truth for the game would be, and how we would avoid data races.

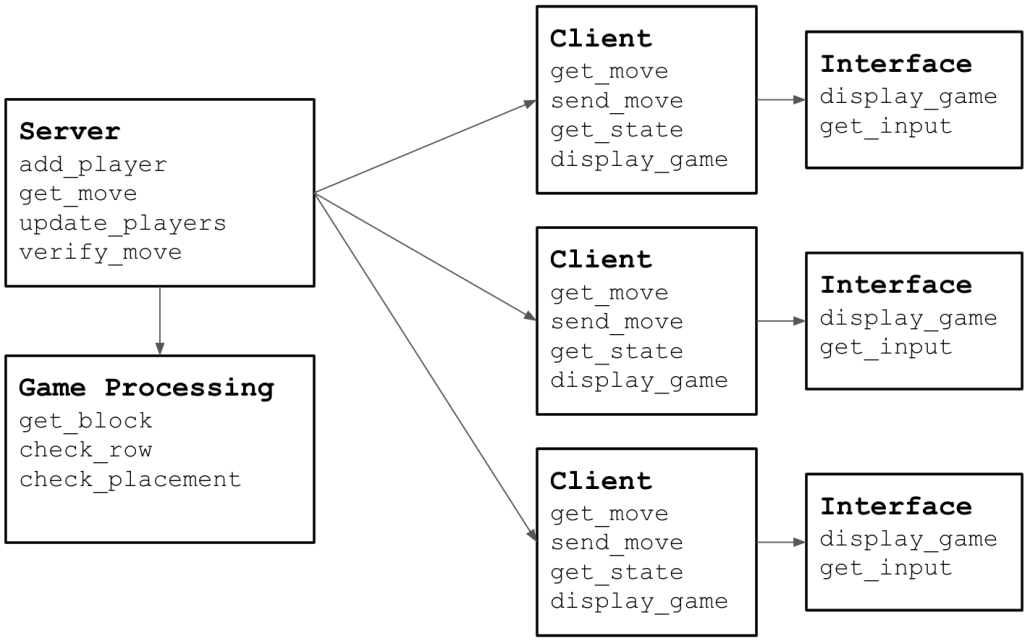
Our first idea was to use the server as a single point of truth, receiving messages from the clients and using the order of those messages to determine how to update the overall game. From there we discussed ways to optimize the weight of the messages by sending only the changes between the users. The second option was to have each client update themselves based on the messages from other users about the actions taken. Finally we discussed the server sending the actions between players as a means of minimizing the messages being sent and keeping the server as a central authority as some degree of a hybrid between the first two options.

Ultimately we decided to pursue the first option, using the server to act as a single point of truth, to simplify the issues of keeping the clients in sync and handling tricky situations where users try to interact with the same location at the same time. This also means the client must not apply their own updates themselves; instead they must wait for the server to send the results of their actions back to them. This means that there will be a longer delay between user input and when the game state actually updates for that user.

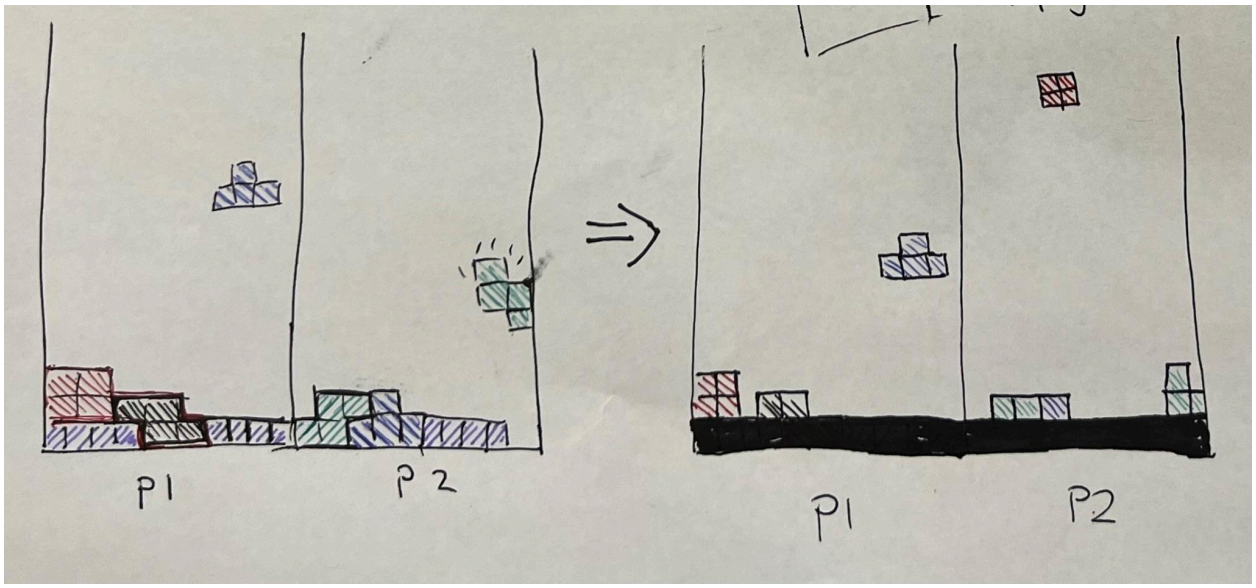
Class Diagram

Below is the class diagram for how the modules interact. The modules include the interface, the clients, the server, and the game-specific calculations. The server acts as a single point of truth, as discussed above, and handles the overall game state. It parses and processes messages from the clients, registering the game state and updating the clients as things change. The game processing module assists the server by checking for cleared lines, generating blocks, tracking scores, and handling all other tasks that are specific to the game of tetris. The game processing module will be spawned for each new game of tetris. As users join the game, new client processes will be created on each user's node. These clients will coordinate the display, receiving messages from the server of the overall game state and updating as necessary. Each client process will also spawn an interface module to handle the display of the game and the user input.

This diagram includes multiple clients as an example, to represent that there will be many players interacting with the server at any given time. Each module also includes some of the functionality we expect to implement.



Object Diagram



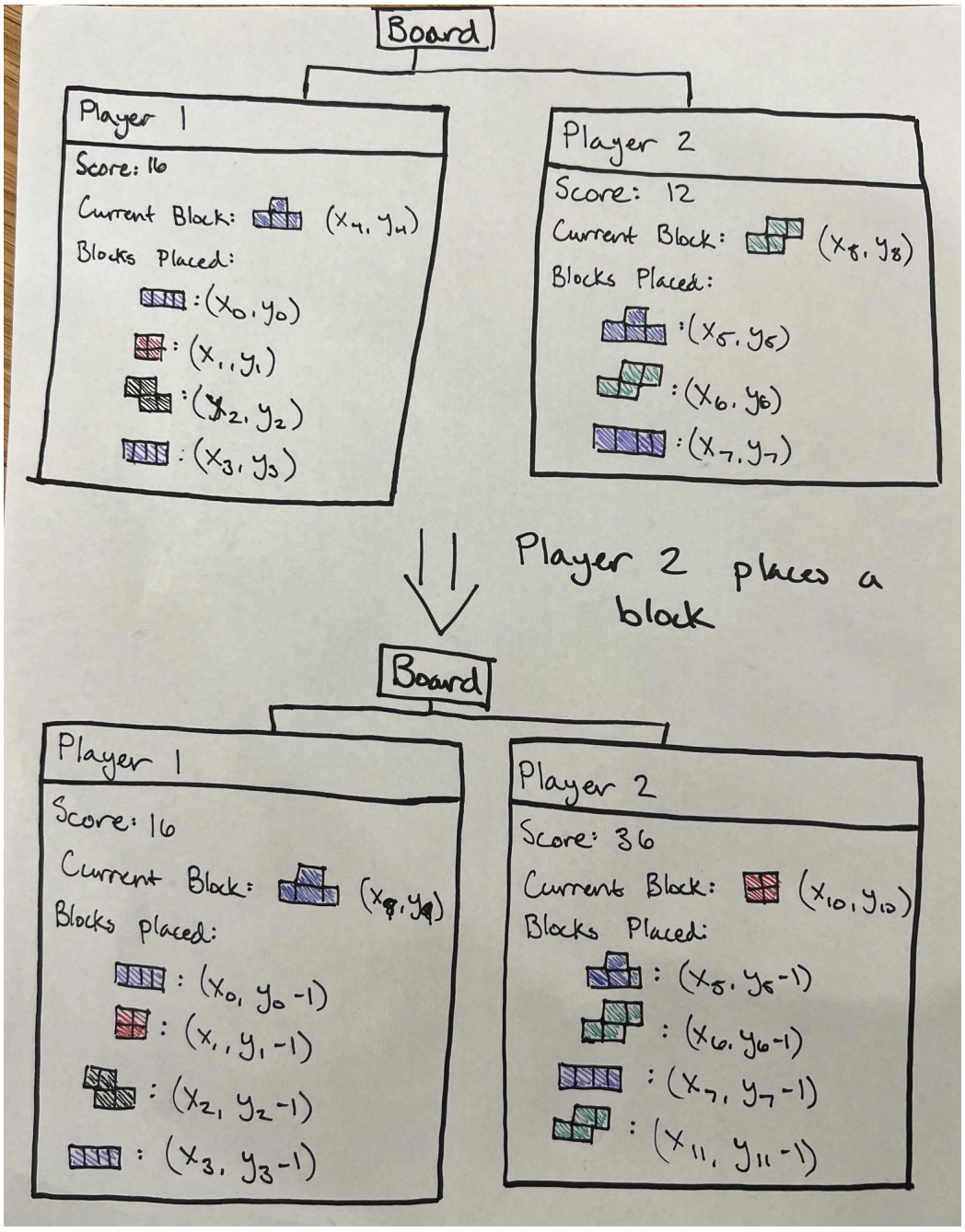
Below we have object diagrams for the clients at two moments within the game. The first moment occurs just after Player 1 places a block. Here is a list of messages that occurred after the block was placed, taking the game to the state represented by the object diagram:

- Client 1 to Server: Player 1 placed block
- Server and Game Processor verify block placement
- Server to all Clients: updated game state, new board, updated scores, and new block for Player 1

- Clients to Interfaces: display new board
- Interface 2 to Client 2: button being pushed, block falling faster

The second moment represented by the object diagrams is just after Player 2 places a block, completing the row. Here is the list of messages that were sent between the moments of the object diagrams:

- Client 2 to Server: Player 2 placed block
- Server and Game Processor verify block placement, recognize line is cleared
- Server to all Clients: updated game state, new board, updated scores, and new block for Player 2
- Clients to Interfaces: display new board



Development Plan

So far we have worked on the initial design for this project, including overall architecture and initial message passing scheme. We have begun to test ncurses and the Erlang project build process and break down roles. We hope to meet every week on Sundays and Thursdays to stay up to date on everyone's work, discuss next steps or blockers, and break up the work moving forward.

Our first steps after this submission will be to finalize our protocols, including specific messages, game state representation, and data structures for the project. After that the work will likely be split between the interface, the client, and the server. We hope that the work will be collaborative and that each team member will understand each piece of the project, but specific roles and responsibilities will allow us to work independently or in pairs. Our overall timeline for the rest of the project to achieve the MVP is below.

Deadline	Goal
Friday, March 29th	Protocols, state representation, data structures
Sunday, March 31st	Ncurses demo
Wednesday, April 3rd	Gen server set up
Friday, April 5th	Tetris visuals
Sunday, April 7th	Game procedures
Friday, April 12th	Single-player tetris
Monday, April 15th	Multiple tetris boards
Monday, April 22nd	Multiplayer tetris!
Monday, April 29th	Final Report

Refined Design

Design Decision Explanation

One of the design decisions we made for this program was regarding the game room of the program and how it would interact with the game. Overall our question was what the source of truth for the game would be, and how we would avoid data races.

Our first idea was to use the game room as a single point of truth, receiving messages from the clients and using the order of those messages to determine how to update the overall game. From there we discussed ways to optimize the weight of the messages by sending only the changes between the users. The second option was to have each client update themselves based on the messages from other users about the actions taken. Finally we discussed the game room sending the actions between players as a means of minimizing the messages being sent and keeping the game room as a central authority as some degree of a hybrid between the first two options.

Ultimately we decided to pursue the first option, using the game room to act as a single point of truth, to simplify the issues of keeping the clients in sync and handling tricky situations where users try to interact with the same location at the same time. This also means the client must not apply their own updates themselves; instead they must wait for the game room to send the results of their actions back to them. This means that there will be a longer delay between user input and when the game state actually updates for that user.

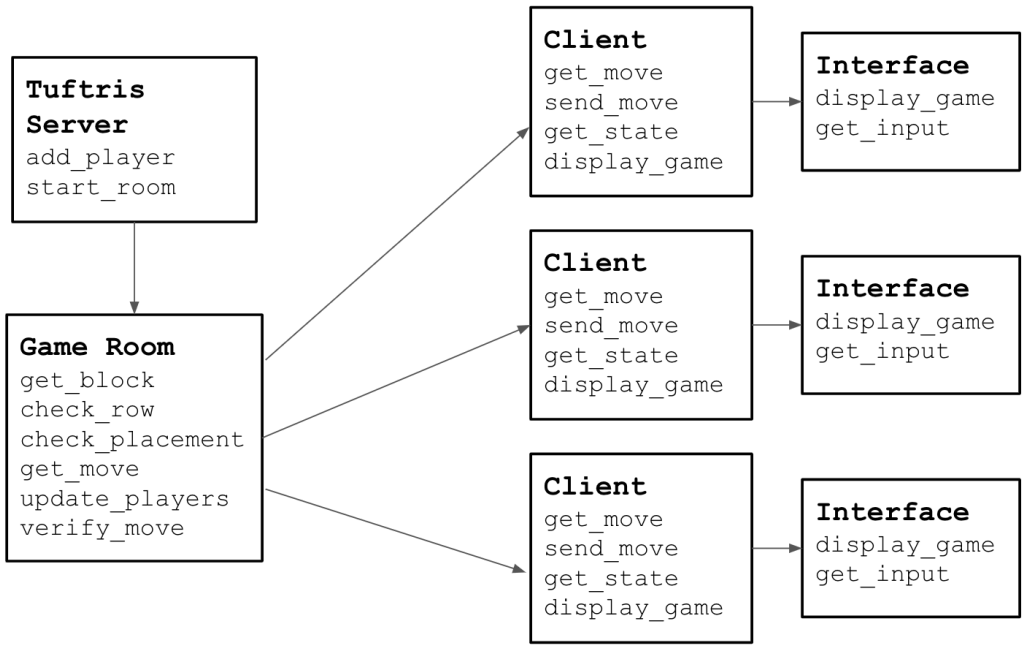
Class Diagram

Below is the class diagram for how the modules interact. The modules include the interface, the clients, the game room, and the server. The game room acts as a single point of truth, as discussed above, and handles the overall game state. It parses and processes messages from the clients, registering the game state and updating the clients as things change. The game room also checks for cleared lines, generates pieces, tracks scores, and handles all other tasks that are specific to the game of tetris (except those handled entirely by individual clients). As users join the game, if they are entering a new room the server will spawn a new game room process; if they are joining an existing room, the server will connect them to the existing room and its players. These clients will coordinate the display, receiving messages from the server of the overall game state and updating as necessary. Each client process will also spawn an interface module to handle the display of the game and the user input.

This diagram includes multiple clients as an example, to represent that there will be many players interacting with the server at any given time. Each module also includes some of the functionality we expect to implement.

Note: this is a change from the previous design. Before we intended to have a server and a game processing module, but as we continued with the project design and architecture we realized that this

method would better support multiple game rooms. The server we had architected before has been enveloped into the game room module with the game processing module, and the new server module now takes the room name input from the user (such as number of players, room name, etc.).



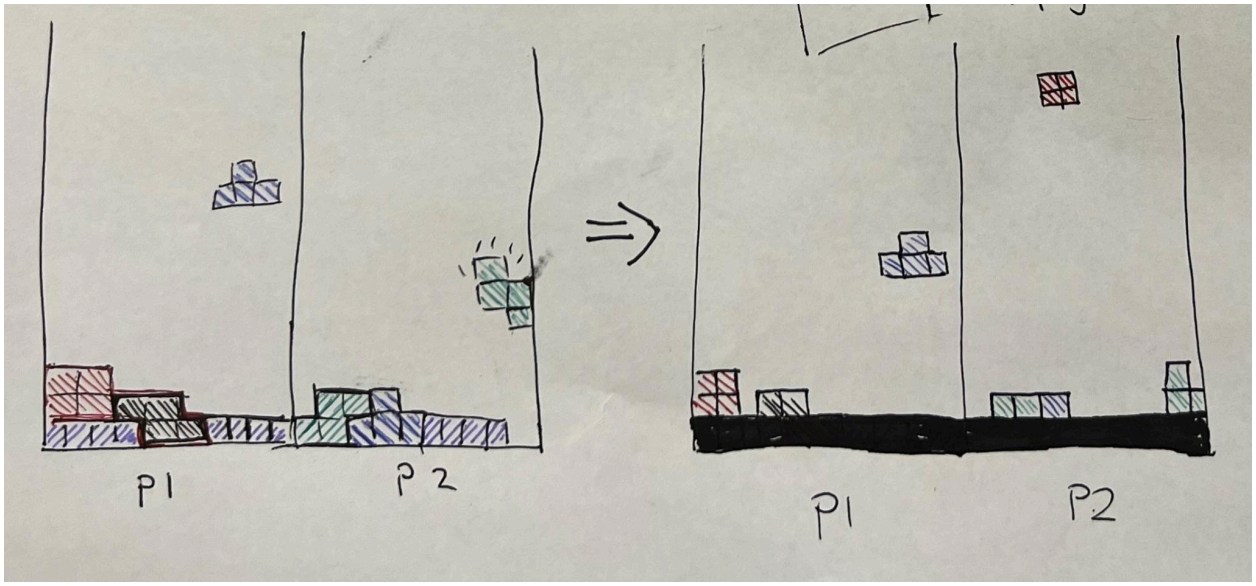
Object Diagram

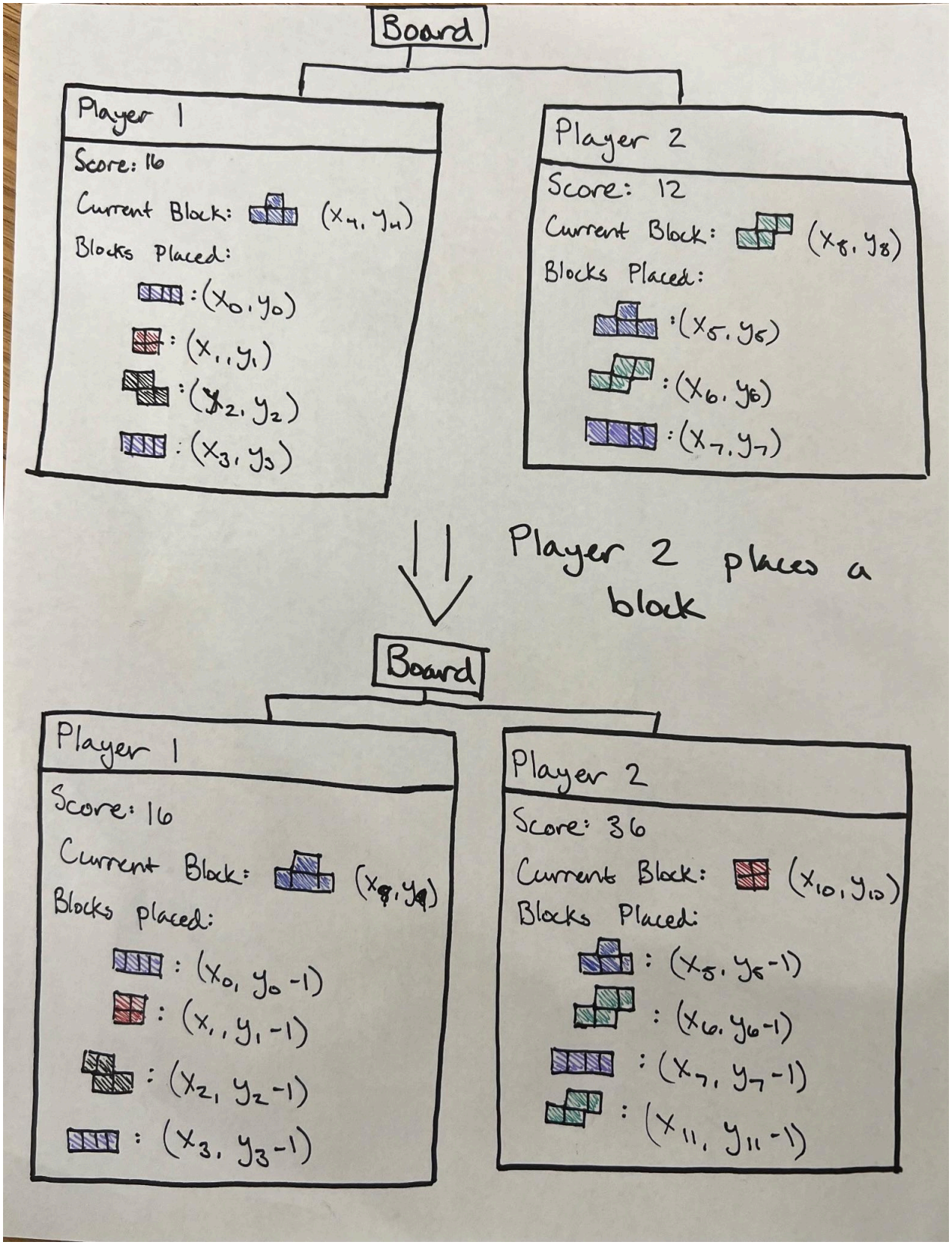
just after Player 1 places a block. Here is a list of messages that occurred after the block was placed, taking the game to the state represented by the object diagram:

- Client 1 to Game Room: Player 1 placed block
- Game Room and Game Processor verify block placement
- Game Room to all Clients: updated game state, new board, updated scores, and new block for Player 1
- Clients to Interfaces: display new board
- Interface 2 to Client 2: button being pushed, block falling faster

The second moment represented by the object diagrams is just after Player 2 places a block, completing the row. Here is the list of messages that were sent between the moments of the object diagrams:

- Client 2 to Game Room: Player 2 placed block
- Game Room and Game Processor verify block placement, recognize line is cleared
- Game Room to all Clients: updated game state, new board, updated scores, and new block for Player 2
- Clients to Interfaces: display new board


P1
P2
P1
P2


Board
Player 1
Score: 16
Current Block: (x_4, y_4)
Blocks Placed:
(x_0, y_0)
(x_1, y_1)
(x_2, y_2)
(x_3, y_3)
Player 2
Score: 12
Current Block: (x_8, y_8)
Blocks Placed:
(x_5, y_5)
(x_6, y_6)
(x_7, y_7)
Player 2 places a block
Board
Player 1
Score: 16
Current Block: (x_8, y_8)
Blocks placed:
$(x_0, y_0 - 1)$
$(x_1, y_1 - 1)$
$(x_2, y_2 - 1)$
$(x_3, y_3 - 1)$
Player 2
Score: 36
Current Block: (x_{10}, y_{10})
Blocks Placed:
$(x_5, y_5 - 1)$
$(x_6, y_6 - 1)$
$(x_7, y_7 - 1)$
$(x_{11}, y_{11} - 1)$

Development Plan

message passing scheme. We have begun to test ncurses and the Erlang project build process and break down roles. We hope to meet every week on Sundays and Thursdays to stay up to date on everyone's work, discuss next steps or blockers, and break up the work moving forward.

Our first steps after this submission will be to finalize our protocols, including specific messages, game state representation, and data structures for the project. After that the work will likely be split between the interface, the client, and the server. We hope that the work will be collaborative and that each team member will understand each piece of the project, but specific roles and responsibilities will allow us to work independently or in pairs. Our overall timeline for the rest of the project to achieve the MVP is below.

Since the submission of the initial design, we have implemented each of the tetris blocks, including movement, color, and rotation, have set up the skeleton of the server, and started working on our game board logic.

Deadline	Goal
Friday, March 29th	Protocols, state representation, data structures
Sunday, March 31st	Ncurses demo
Wednesday, April 3rd	Gen server set up
Saturday, April 6th	Tetris visuals
Thursday, April 11th	Game procedures
Sunday, April 14th	Single-player tetris
Wednesday, April 17th	Multiple tetris boards
Monday, April 22nd	Multiplayer tetris!
Monday, April 29th	Final Report

Abstraction and Language

In designing Tuftris we have developed language to understand different aspects of our program, these are our main abstractions.

Tetromino: shape made from 4 blocks.

Board: A data structure that represents the tetris game state for each player. Inherently has a visual aspect that is rendered based on the state of the data structure and falling **pieces.**

Piece: A tuple that represents the tetromino shapes of Tetris. They are identified by the tuple {Type, Rotation, Center, Cells}. These **pieces** have several different types, which are listed below, the rotation is a qualifier 0-3, center is the location of center of rotation, and **cells** are defined below.

Cell: A single square “tile” on the board. The **board** is a grid of cells.

T: The T-shaped tetromino, traditionally purple.

Square: The square-shaped tetromino traditionally yellow.

Left: The L-shaped tetromino, traditionally orange.

Right: The flipped L-shaped tetromino, traditionally blue.

Zigz: The Z-shaped tetromino, traditionally red.

Zags: The S-shaped tetromino, traditionally green.

Line: The line-shaped tetromino, traditionally blue.