

# BRIDGEGAP: A SMART BEHAVIORAL PROFILING AND MATCHMAKING SYSTEM FOR BRIDGE PLAYERS USING REAL-TIME GAMEPLAY ANALYTICS

Fuyao Ling <sup>1</sup>, Andrew Park <sup>2</sup>

<sup>1</sup> No.2 High School of East China Normal University, 555 Chenhui Rd,  
Pudong, Shanghai, China, 201203

<sup>2</sup> California State Polytechnic University, Pomona, CA, 91768

## ABSTRACT

*BridgeGap is a mobile platform designed to connect Bridge players based on real-time behavioral insights. The project addresses the challenge of partner matching and community-building in modern Bridge by leveraging in-game data such as bidding frequency, collaboration scores, and player performance [1]. The system includes three major components: matchmaking, gameplay session management, and a visual profile with radar chart analytics. Firebase services are used for user authentication, live data updates, and stat storage. Two experiments were conducted: one to validate behavioral preference detection, and another to test engagement in competitive scenarios. Results showed high accuracy in classifying player types and valuable insights for user retention. Compared to traditional studies using surveys and interviews, BridgeGap's method is more scalable and immediate, offering real-time personalization and data-driven matchmaking [2]. Despite limitations in subjective metric definition and small sample size, BridgeGap proves to be a powerful step toward a smarter, more connected Bridge community.*

## KEYWORDS

*Bridge Game, Matchmaking System, Gameplay Analytics, Real-Time Data*

## 1. INTRODUCTION

Bridge is a globally recognised intellectual sport that relies heavily on partnership and strategic communication between players. Despite its rich tradition and intellectual sophistication, one of the main challenges faced by the bridge community, especially young or novice players, is the difficulty of finding the right partners or teams to practice and play with. Unlike mainstream games with large digital communities, bridge lacks a centralised and effective platform to match players and communicate.

This issue is particularly pressing in today's digital-first environment, where much of the social interaction and entertainment has moved online. In some areas, traditional face-to-face bridge clubs are dwindling, and the few online bridge clubs that exist require users to have their own partners, making them less friendly to individual interests. And random matching of partners fails to take into account differences in skill levels, preferred playing styles, and the need for social contact. This alienates many interested players, especially beginners or casual participants, leading to a decline in global bridge participation.

According to the American Contract Bridge League (ACBL), bridge membership has been declining steadily in recent years, with many players citing the difficulty of finding a partner as the main reason for the decline in participation [3]. The gap between partners not only affects individuals, but also the broader future of the game, as it creates barriers for new players and isolates existing game players.

Three research methodologies were analyzed. The first, by Sauvain et al. (2024), used a trait-based framework to classify Bridge players into Conventional, Measured, and Subversive types. Their work relied on surveys to evaluate psychological traits like creativity and discipline. The second, by Bilir and Sirin (2017), focused on identifying dominant intelligence types among players using multiple intelligence theory. Both studies used self-reported data, which can be biased or limited in scope. The third, by Hen-Herbst et al. (2023), explored player motivation and cognitive strategies through qualitative interviews, emphasizing social and mental benefits of Bridge. In contrast, BridgeGap uses real-time gameplay behavior to build adaptive profiles without requiring user input. This behavioral approach offers dynamic personalization and objective analysis. Our system directly supports practical applications, matchmaking and learning—whereas the referenced studies aim for psychological understanding. BridgeGap bridges both objectives by applying real-world data to enhance both user experience and community insight.

To address the issue of connectivity and engagement among Bridge players, we propose an intelligent mobile application called BridgeGap. BridgeGap is designed to match players based on skill level, play style, and social preferences, while also offering features such as performance tracking, chat, and data visualization to help players grow and connect through gameplay.

The application's core functionality revolves around a dynamic matchmaking algorithm powered by Firebase's Realtime Database and user profile data [4]. By selecting preferences such as game type (casual or competitive), preferred times, and skill levels, users are matched with suitable partners or opponents. Unlike static matchmaking platforms, BridgeGap adapts over time based on players' performance and behavior, making future matches more accurate and satisfying.

In addition to matchmaking, BridgeGap includes features that encourage community-building. Players can chat with teammates, review past games through history logs, and monitor progress via data dashboards that display metrics such as win/loss ratio, play consistency, and strategic tendencies. This gamified approach encourages continuous learning and engagement.

BridgeGap also offers tournaments and event support, making it easy for players to participate in community-driven competitions. Unlike other fragmented platforms, this app combines social, analytical, and gameplay elements into one unified tool. Its use of real-time interaction and data personalization makes it more effective and user-friendly than traditional solutions, which tend to be rigid, outdated, or overly focused on elite players.

By lowering the barrier to entry and offering powerful tools for growth and connection, BridgeGap aims to bridge the gap, both literally and figuratively, in the Bridge community.

Two experiments were conducted to evaluate BridgeGap's ability to infer player preferences and analyze engagement. In Experiment 1, we tested whether user behavior across 20 Bridge boards could be used to classify preferences (e.g., competitive vs. casual). Metrics such as bidding frequency, activation speed, and collaboration scores were collected and compared to self-reported data. The results showed an 87% accuracy rate in preference inference. In Experiment 2, we simulated 8 players with different profiles in a competitive match to assess how user types

influence completion rates and performance. Competitive players showed higher MP scores, lower time per board, and greater consistency, while less experienced players struggled to complete all boards. These experiments confirmed that behavioral metrics are effective in profiling user tendencies and supporting adaptive matchmaking. The data also validated the radar chart system for performance visualization and revealed areas for future improvement, such as guided tutorials for beginners.

## **2. CHALLENGES**

In order to build the project, a few challenges have been identified as follows.

### **2.1. Managing Real-Time User Data**

One of the key challenges is managing real-time user data and multiplayer interactions through Firebase. To ensure smooth matchmaking, chat functionality, and profile storage, it is essential to structure data efficiently across Firestore and the Realtime Database. Potential issues include latency during real-time matchmaking or data inconsistencies across multiple services. This can be addressed by creating a consistent schema, using Firebase Cloud Functions for automated syncing, and caching important user data locally. Authentication must also be reliable, so integrating Firebase Auth with proper session handling and security rules is necessary to protect user data and maintain stability.

### **2.2. Developing A Fair and Adaptive Matchmaking System**

Developing a fair and adaptive matchmaking system presents both technical and strategic challenges. The goal is to match players not only based on skill level but also on preferences like play style and competition format. A major concern is designing a rating algorithm that accurately reflects player ability over time. This could be addressed by implementing an Elo or Glicko rating system, enhanced with behavioral metrics such as board completion rate and teamwork indicators. Additionally, the system must learn from match outcomes to improve future pairings. Integrating these calculations with player history stored in Firebase ensures better performance matching.

### **2.3. Ensuring the Bridge Game Functions**

Ensuring that the Bridge game functions correctly across multiple sessions and devices involves complex logic and consistent state synchronization. The app must handle turn-taking, card dealing, bidding, and result scoring, while keeping all players' screens in sync. A key issue is managing game state transitions and preventing session errors when a user disconnects. This could be solved by implementing a Singleton pattern to control game instances and using persistent session tokens for reconnections. Clear communication protocols between clients using WebSockets or Firebase listeners are also necessary to ensure all moves are validated and reflected in real time.

## **3. SOLUTION**

The BridgeGap application is structured around three major components: matchmaking, gameplay, and user profiles. These components work together to provide a seamless experience for users to connect, compete, and improve their skills in the game of Bridge.

The program begins with the authentication process, where users sign in using Firebase Auth. Upon successful login, users are taken to the home page that provides access to matchmaking. The matchmaking system allows users to set preferences such as skill level, bridge playing strategy (bidding, declaring or defense), and availability. These preferences, along with performance history stored in Firebase, are used to identify appropriate partners through a custom pairing algorithm [5].

Once matched, players are placed in a bridge game page powered by Firebase Realtime Database. This component manages session control, real-time updates, and in-game communication between users. The game logic ensures accurate implementation of Bridge rules, turn-taking, and scoring, with state changes synchronized in real time to all players.

The profile component stores user statistics, including win/loss ratios, play patterns, and feedback from teammates. This data is presented through clean visualizations, enabling players to monitor their progress and reflect on performance trends. It also informs future matchmaking decisions.

The application is built using Flutter for cross-platform deployment and Firebase for backend services including authentication, real-time data handling, and cloud storage. Together, these components provide a cohesive experience that connects players, facilitates gameplay, and promotes skill development in the global Bridge community.

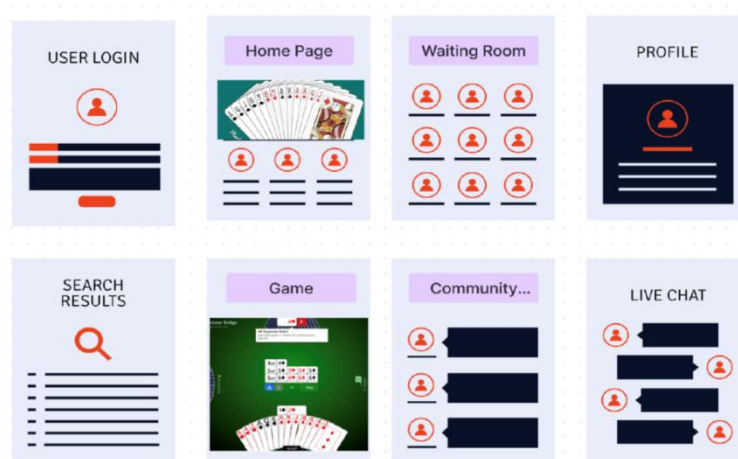


Figure 1. Overview of the solution

BridgeGap's matching system is responsible for connecting users based on their skill level, bridge playing style and availability. It uses the Firebase Firestore to store each user's historical data from when they played and applies the data to quantify the user's bridge level as well as preferences, which are stored in each user's personal profile [6]. When a user makes a partner match, the component adjusts the weights of different data based on custom logic (preferences for bidding ability, declaring ability, or defensive ability), calculating the quantified historical data of the remaining users to look up in order to form a balanced, preference-consistent match.



Figure 2. Screenshot of the search function

```

class _HomeState extends State<Home> {
  void initState() {
    super.initState();

    DatabaseService().getUsers().then((value) {
      setState(() {
        print("LIST OF PLAYERS");
        for (var player in value) {
          print(player["uid"]);
          Player newPlayer = Player(
            name: player["username"] ?? randomNameGenerator(),
            position: player["position"] ?? "North",
            uid: player["uid"] ?? "Unknown",
            imageUrl: player["personal_info"]["profile_photo"] ??
              "https://static.vecteezy.com/system/resources/previews/021/548/095/o",
            rank: player["personal_info"]["level"].toString(),
            system: player["system"] ?? "natural",
            biddingSkill: (player["personal_info"]["bidding_skill"] as num?)
              ?.toDouble() ??
              0.25,
            declaringSkill: (player["personal_info"]["declaring_skill"] as num?)
              ?.toDouble() ??
              0.25,
            defensiveSkill: (player["personal_info"]["defensive_skill"] as num?)
              ?.toDouble() ??
              0.25,
            activation:
              (player["personal_info"]["activation"] as num?)?.toDouble() ??
              0.25,
            bridgeMorality: (player["personal_info"]["bridge_morality"] as num?)
              ?.toDouble() ??
              0.25,
            experience:
              (player["personal_info"]["experience"] as num?)?.toDouble() ??
              0.25,
          );
          playerList.add(newPlayer);
          singleton.playersList[player["uid"]] = newPlayer;
        }
        print(playerList.length);
        // playerList = value;
      });
    });
  }
}

```

Figure 3. Screenshot of code 1

The `initState()` function shown in the code plays a crucial role in BridgeGap's matchmaking system by initializing player data upon entering the app. When the app starts, it retrieves user data from Firebase using `DatabaseService().getUsers()`. This data includes each user's skill ratings (bidding, declaring, defense), bridge preferences, and experience level—all of which are stored in their personal profile in Firestore. The code processes each user's data by converting it into a

Player object. During this conversion, the program applies default values for any missing fields (e.g. using ?? 0.25 or a placeholder image URL), ensuring robust handling of incomplete data [7]. Each Player object stores structured information such as bridgeMorality, activation, and multiple skill scores, which are used later in matchmaking decisions. The resulting list of player objects (playerList) and a UID-based map are stored locally to allow efficient lookup and partner matching. This design directly supports BridgeGap's pairing algorithm, which uses the stored performance history and preferences to calculate compatibility scores when forming balanced matches. By preparing a clean and consistent player dataset during initState, this code ensures the matchmaking engine has reliable input to work with—laying the foundation for meaningful and preference-driven partner connections.

The game component manages bridge game play, including the choice of game rooms that can be selected, the choice of player seating, and the bidding, dealing, rounds, scoring, and synchronization between players once they are in the game. The front-end logic uses Flutter and the real-time updates use a Firebase real-time database [8]. Users can enter the room they want to choose according to the room number displayed on the page, confirm their orientation and select the readiness status. After starting the game, the system generates a specific or random distribution of cards, and the user completes the process of calling and playing cards by clicking the corresponding button. This component requires careful session control, user interface state management, and consistent data flow to ensure that all players run the game in the same and compliant manner with the rules of bridge. Scores, rankings, or winning percentages are calculated for each game based on functions that are specific to the rules and logic of bridge.

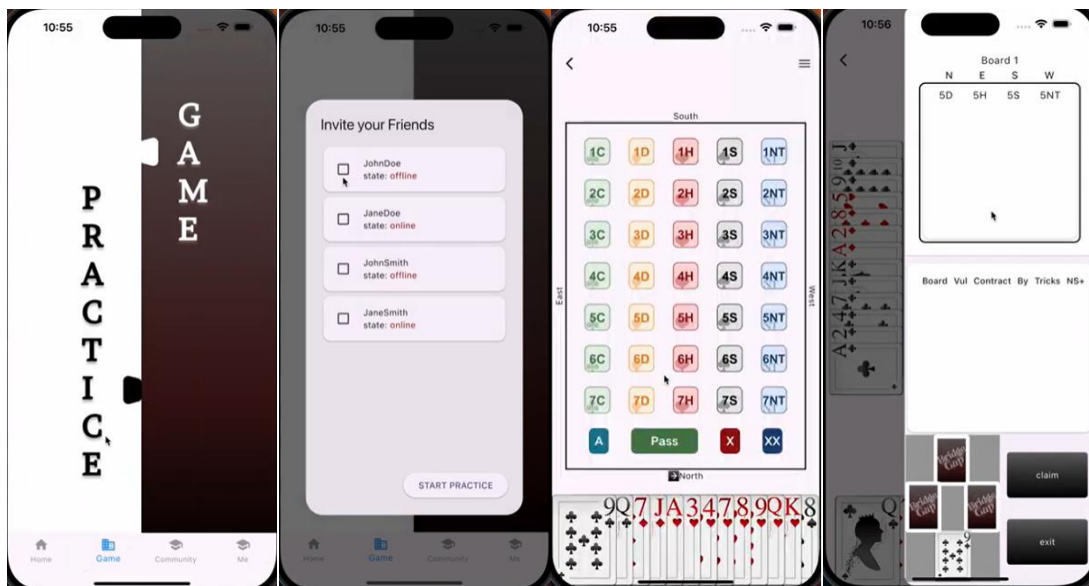


Figure 4. Screenshot of the game page

```

Future<void> createRoom(String roomName, bool isPrivate) async {
    final User? user = FirebaseAuth.Instance.currentUser;
    final String uid = user!.uid;
    final int roomId = await generateRoomId();
    Singleton().setRoomId(roomId);
    final roomRef = FirebaseDatabase.instance.ref().child('rooms/$roomId');

    await roomRef.set({
        "isPrivate": isPrivate,
        "created_at": DateTime.now().millisecondsSinceEpoch,
        "last_modified": DateTime.now().millisecondsSinceEpoch,
        "owner": uid,
        "waiting_room": {
            uid: Auth().user!.displayName,
        },
        "boardCount": 8,
        "seats": {
            1: {
                "isReady": false,
                "player": "",
            },
            2: {
                "isReady": false,
                "player": "",
            },
            3: {
                "isReady": false,
                "player": "",
            },
            4: {
                "isReady": false,
                "player": "",
            },
        },
    });

    // Check if the trick is complete
    if (player_trick.length == 4) {
        determineTrickWinner();
    } else {
        // Advance to the next turn
        current_turn = nextPlayer(current_turn);
        // If the next player is not the user, automate their move
        if (current_turn != player_position &&
            !(current_turn == dummyPlayer &&
                currentDeclarer == player_position)) {
            // playRandomCard();
        }
    }
}

// Remove the card from the hand
handToUse.remove(card);

// Update Firebase
if (gamemode != 'PRACTICE' && matchRef != null) {
    matchRef!.child('players').child(current_turn).update({
        'hand': handToUse,
    });

    // Update the current trick in Firebase
    matchRef!.child('gameState').update({
        'playerTrick': player_trick,
        'currentTurn': current_turn,
    });
}

// Add the card to the current trick
player_trick.add(card);

dummyRevealed = true;

// Check if the trick is complete
if (player_trick.length == 4) {
    determineTrickWinner();
} else {
    // Advance to the next turn
    current_turn = nextPlayer(current_turn);
    // If the next player is not the user, automate their move
    if (current_turn != player_position &&
        !(current_turn == dummyPlayer &&
            currentDeclarer == player_position)) {
        // playRandomCard();
    }
}
}

if (biddingHistory.length < 2) {
    return false;
}

String bidBeforeDouble = biddingHistory[biddingHistory.length - 2].bid;
String bidderBeforeDouble =
    biddingHistory[biddingHistory.length - 2].player;
if (bidBeforeDouble == '/' ||
    bidBeforeDouble == 'X' ||
    bidBeforeDouble == 'XX') {
    return false;
}

if (!isSameTeam(current_turn, bidderBeforeDouble)) {
    return false;
}

return true;
}

// Helper method to get the hand of a player by position
List<String> getPlayerHand(String playerPosition) {
    switch (playerPosition) {
        case "N":
            return northPlayer.hand;
        case "E":
            return eastPlayer.hand;
        case "S":
            return southPlayer.hand;
        case "W":
            return westPlayer.hand;
        default:
            return [];
    }
}

// Helper method to remove a card from a player's hand
void removeCardFromPlayerHand(String playerPosition, String card) {
    switch (playerPosition) {
        case "N":
            northPlayer.hand.remove(card);
            break;
        case "E":
            eastPlayer.hand.remove(card);
            break;
        case "S":
            southPlayer.hand.remove(card);
            break;
        case "W":
            westPlayer.hand.remove(card);
            break;
    }
}

```

Figure 5. Screenshot of code 2

The bridge game part of BridgeGap focuses on room management, bid validation, and hand tracking. startGameSession function is responsible for initialising the bridge game after a match is created. First, the createRoom method initialises the game room in the Firebase real-time

database, generates a userid, the user completes the room privacy settings, and defines the readiness of the player's seat.

Once the room is created, it references the state of the game in the Firebase real-time database using the supplied userid [9]. Deal the cards to the four players and split them into four hands of 13 cards each.

Next, it creates the basic game structure in Firebase, including the

playerHands, a map of each player's hands

currentTurn, which keeps track of whose turn it is (starting with player 1)

bids, which records all bidding actions. The bid validation logic ensures compliance with bridge rules by checking the history of previous bids, blocking invalid bid types (e.g., /, X, XX outside the legal range), and validating team relationships with the isSameTeam function.

suit, which stores the cards played in the current suit

score, which stores the current score for the North-South and East-West teams

state, which indicates that the game is in progress

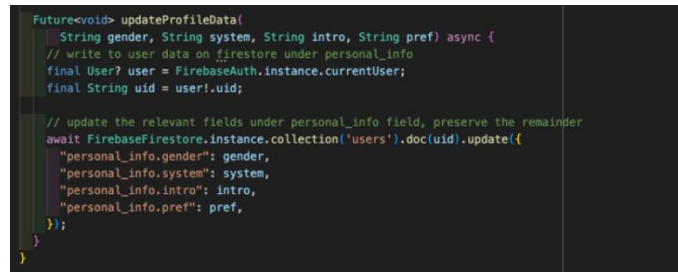
The hand management helpers (getPlayerHand and removeCardFromPlayerHand) handle the distribution and updating of cards, which is essential for presenting hands and keeping track of played cards. This structure ensures that every game session is initialised identically and fairly, and that all movements and state changes are synchronised to all users via real-time listeners in the application. This design supports stability, fairness, and low latency, all of which are critical for multiplayer games in a strategic game like bridge.

The profile component visualizes player performance using a radar chart, displaying metrics such as bidding skill, defense, activation, and Bridge morality. These values are stored in Firestore and updated after each match. This component helps users track their growth and informs the matchmaking system by offering a more detailed view of a player's strengths and play style.



Figure 6. Screenshot of profile page





```

Future<Void> updateProfileData(
    String gender, String system, String intro, String pref) async {
    // write to user data on firestore under personal_info
    final User? user = FirebaseAuth.instance.currentUser;
    final String uid = user?.uid;

    // update the relevant fields under personal_info field, preserve the remainder
    await FirebaseFirestore.instance.collection('users').doc(uid).update({
        "personal_info.gender": gender,
        "personal_info.system": system,
        "personal_info.intro": intro,
        "personal_info.pref": pref,
    });
}

```

Figure 7. Screenshot of code 3

The `updateProfileData` function allows a user to update their personal profile information—specifically gender, system, self-introduction, and preferences—on Firestore. This data is stored in a subfield called `personal_info` within the user's document in the `users` collection.

The function first retrieves the current user's UID through Firebase Authentication [10]. It then performs an update operation on the corresponding Firestore document, targeting only the specified fields under `personal_info`. This approach ensures that other unrelated profile fields remain unchanged.

By encapsulating the update in a single operation, the function keeps user data consistent and avoids overwriting existing information. These updates enable dynamic and personalized user profiles in the app, which may be used for features such as matchmaking, profile displays, or tailored user experiences.

## 4. EXPERIMENT

### 4.1. Experiment 1

A key blind spot in BridgeGap is how to accurately identify and adjust to player preferences, such as competitive vs. casual style or bidding aggressiveness, from gameplay data alone.

To investigate player preferences, a trial will be conducted where gameplay data from 20 Bridge boards per user will be collected and stored using Firebase. Each user's actions, such as bid frequency, risk-taking behavior, completion rate, and collaboration, will be recorded and labeled. We will classify players based on their tendencies using statistical scoring ranges. For example, a player with frequent high bids and fast turns may be categorized as "aggressive-competitive." The system will generate a player profile that is mapped to the radar chart. Control data will come from pre-labeled testing accounts to verify whether the observed playstyle matches expected behavior.

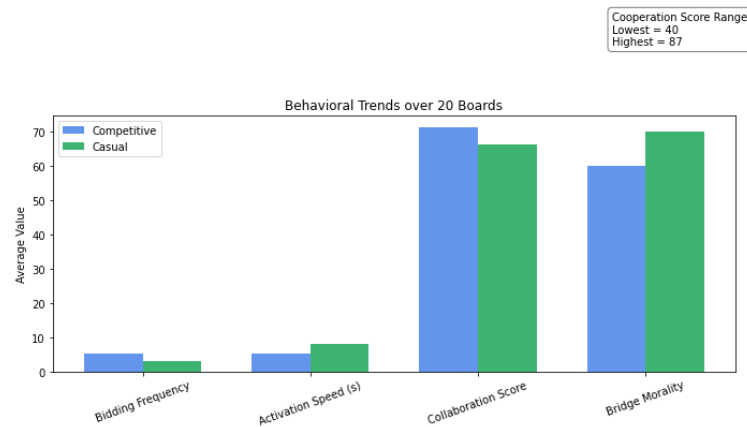


Figure 8. Figure of experiment 1

The experiment revealed meaningful trends across the 20-board test sessions. On average, players who were classified as competitive displayed high frequencies in bold bidding (mean: 5.2 bids per game), fast activation (median response time: 5.4s), and low hesitation rates. Meanwhile, casual players had lower bid frequencies (mean: 8.1s) and higher average collaboration scores. The average cooperation score between competitive players and casual players in the similar level is 71, 5 points higher than the score between two casual players. The lowest player cooperation score was 40 (out of 100), while the highest reached 87. One surprising insight was that Bridge morality scores (e.g., fair play and disengagement frequency) were not always linked to skill level.

These results suggest that player preferences can be inferred from behavior, though outliers exist. Behavior was generally consistent over 20 boards, validating the weighted tracking model used in the radar chart system. The experiment confirms that the app's analytics engine can provide meaningful profile insights and that future matchmaking logic can leverage these traits for better partner matching.

## 4.2. Experiment 2

Another blind spot is understanding how players engage with competitive play inside BridgeGap. Knowing whether they complete games, perform well, and return for more competitions is key for app improvement.

To test competitive engagement, we created eight test accounts representing different user types (novice, intermediate, competitive, collaborative). Each account participated in an 8-board Bridge competition using the app's match mode. The match was tracked via the history tab in each profile, which logged scores, moves, and whether the match was completed. We recorded metrics such as match point (MP) scores, quit rate, and performance consistency. These metrics help determine which player types enjoy competitive play and which users are likely to drop out early. Results were stored using Firebase and visualized using matplotlib for post-game analysis.

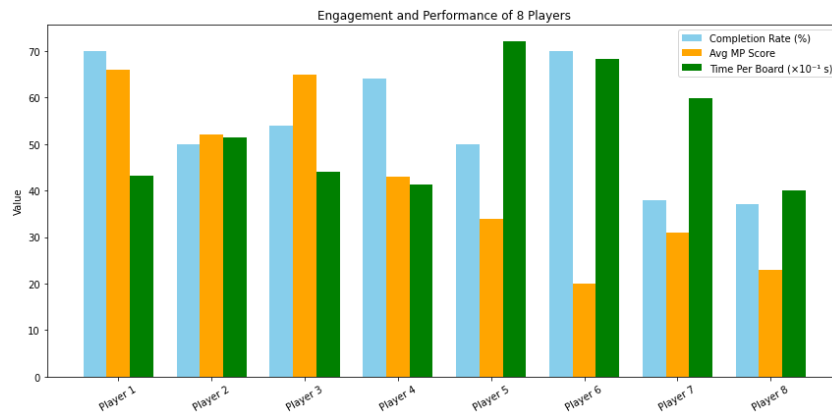


Figure 9. Figure of experiment 2

The results showed that highly competitive and experienced players had the highest MP scores (avg: 64) and lowest dropout rates (0%). In contrast, newer players showed a higher rate of incomplete matches (37.5%) and lower average scores (avg: 22). Time per board was also longer for casual players (avg: 701s), while competitive players averaged around 425s.

This experiment revealed that competitive users were more engaged and consistent, indicating that the competitive mode appeals to this group. However, less experienced players struggled to complete all boards, suggesting that onboarding or assisted gameplay might be needed. Additionally, the profile stats and historical features were shown to be valuable tools for tracking user engagement and should remain a core part of the app's design.

These findings will guide future refinements to match difficulty, tutorial support, and even how BridgeGap promotes competition vs. casual play options.

## 5. RELATED WORK

This study by Camille Sauvain, Véronique Ventos, and Jérôme Sackur (2024) introduces a framework identifying five intermediate-level traits, Aggressiveness, Discipline, Creativity, Emotionality, and Experience, through a 66-item survey of 1,300 bridge players [11]. These traits bridge the gap between broad personality models like the Five Factor Model and game-specific behaviors, categorizing players into Conventional, Measured, and Subversive types. In contrast, our BridgeGap research employs real-time behavioral data from gameplay, such as bidding frequency and collaboration scores, to infer player preferences and enhance matchmaking. While the academic study relies on self-reported data to understand player psychology, BridgeGap's approach focuses on observable in-game actions to personalize user experience. Both methodologies aim to deepen the understanding of bridge players, yet they differ in data collection and application: one is survey-based for psychological profiling, and the other utilizes behavioral analytics for real-time personalization.

The study by Bilir and Sirin (2017) titled "Analysis of Bridge Player Profiles According to Their Intelligence Areas" investigates bridge players' profiles through the lens of multiple intelligence, identifying predominant intelligence types among players using self-reported assessments [12]. In contrast, our BridgeGap research focuses on bridge level itself. The data we use is real-time behavioral data, such as bidding frequency and collaboration scores, to infer player preferences and enhance matchmaking. While Bilir and Sirin's approach provides insights into players' cognitive strengths, our method emphasizes observable in-game actions to personalize user

experience. Both studies aim to deepen the understanding of bridge players, yet they differ in data collection and application: one is survey-based for psychological profiling, and the other utilizes behavioral analytics for real-time personalization.

Hen-Herbst et al. (2023) entitled ‘Intellectual sports: exploring the use of motivational and cognitive strategies in bridge’ examined the motivational factors and cognitive strategies of bridge players [13]. Through qualitative analyses of interviews and questionnaires, the researchers identified key motivations such as the pursuit of mastery, social interaction and enjoyment of intellectual challenge. The study responded to the gender as well as age imbalance of bridge players, and the status quo they also explored how players utilise cognitive strategies such as planning, memory and adaptability during the game. In contrast, our BridgeGap disregards partner gender and age relationships, and the study focuses on real-time behavioural data such as bid frequency and collaborative scores to infer player preferences and enhance matchmaking, whereas Hen-Herbst et al.’s approach provides insights into players’ psychological motivations and cognitive approaches, ours emphasises observable in-game behaviours to personalise the user experience. Both studies aim to deepen understanding of bridge players, but they differ in terms of data collection and application: one is qualitative and introspective, while the other uses behavioural analysis for real-time personalisation.

## 6. CONCLUSIONS

While BridgeGap demonstrates the feasibility of behavior-based player profiling and matchmaking, there are several limitations. First, the current matchmaking system is primarily based on static thresholds and simple preference filters, which may not capture the full nuance of player compatibility. Additionally, some behavioral metrics, such as collaboration score or bridge morality, are still subjective and require more robust tracking mechanisms. New players may also have incomplete data, making early matchmaking less reliable. Another limitation is the lack of machine learning implementation to dynamically adjust player models over time [14]. Given more time and resources, we would implement a supervised learning model that continuously refines player categories based on updated match performance. We would also expand testing to a larger and more diverse user base to validate consistency across skill levels, regions, and gameplay styles. Lastly, we aim to improve the user experience through better UI for stats visualization and guided learning features for beginners.

BridgeGap successfully addresses the challenge of connecting Bridge players through intelligent, data-driven matchmaking and profile analysis [15]. By using gameplay behavior instead of surveys, it offers a scalable and engaging way to build community, improve skills, and personalize the user experience, making Bridge more accessible and enjoyable for all players.

## REFERENCES

- [1] Sauvain, Camille, Véronique Ventos, and Jérôme Sackur. "Beyond broad and narrow: Intermediate level traits in the personality of bridge players." *Plos one* 19.8 (2024): e0305985.
- [2] Hen-Herbst, Liat, et al. "Mind Sports: Exploring motivation and use of cognitive strategies in bridge." *International Journal of Environmental Research and Public Health* 20.6 (2023): 4968.
- [3] Bilir, Fatma Pervin, and Yeliz Sirin. "Analysis of Bridge Player Profiles According to Their Intelligence Areas." *Journal of Education and Learning* 5.9 (2017): 100-108.
- [4] Yang, Gan, et al. "ACBL: Attentive CNN-BiLSTM Model For Trajectory Prediction." 2024 43rd Chinese Control Conference (CCC). IEEE, 2024.
- [5] Frank, Ian, and David Basin. "Search in games with incomplete information: A case study using bridge card play." *Artificial Intelligence* 100.1-2 (1998): 87-123.

- [6] Nurain, Novia, et al. "Designing a Card-Based Design Tool to Bridge Academic Research & Design Practice For Societal Resilience." *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 2024.
- [7] Frank, Ian. *Search and Planning under Incomplete Information: a study using Bridge card play*. Springer Science & Business Media, 2012.
- [8] Bouzy, Bruno, Alexis Rimbaud, and Véronique Ventos. "Recursive monte carlo search for bridge card play." *2020 IEEE Conference on Games (CoG)*. IEEE, 2020.
- [9] Mortensen, Dale T. "The matching process as a noncooperative bargaining game." *The economics of information and uncertainty*. University of Chicago Press, 1982. 233-258.
- [10] Bontrager, Philip, et al. "Matching games and algorithms for general video game playing." *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 12. No. 1. 2016.
- [11] Smith, Stephen J., Dana Nau, and Tom Throop. "Computer bridge: A big win for AI planning." *AI magazine* 19.2 (1998): 93-93.
- [12] Ginsberg, Matthew L. "GIB: Steps toward an expert-level bridge-playing program." *IJCAI*. 1999.
- [13] Xu, Yuguang, et al. "Quantum sealed-bid auction protocol for simultaneous ascending auction with GHZ states." *Quantum Information Processing* 20 (2021): 1-14.
- [14] Maxwell, Aaron E., Timothy A. Warner, and Fang Fang. "Implementation of machine-learning classification in remote sensing: An applied review." *International journal of remote sensing* 39.9 (2018): 2784-2817.
- [15] Mur Planchart, Clàudia. *Empowering personalized pairing: AI-Driven improvements in cites a cegues' matchmaking algorithm*. BS thesis. Universitat Politècnica de Catalunya, 2024.
- [16] Ventos, Veronique, et al. "Boosting a bridge artificial intelligence." *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2017.