PlayPal - Game Storage and Retrieval with Redis

Rachel Cassway, Tijana Cosic, Brady Duncan, Joshua Peirce, Rachel Utama

Northeastern University, Boston, MA, USA

Abstract

PlayPal is a game application backed by Redis for the user and game storage and retrieval. This initial rendition of PlayPal runs a Tic-Tac-Toe game that a user can play live against the computer. Our user-friendly interface allows anyone to use the PlayPal app very easily. Another goal we succeeded in was writing reusable code that allows us to load new games to PlayPal, such as Checkers or Connect 4, in future PlayPal releases. We also worked to isolate the Redis API which would allow us to swap in different APIs made for different database environments.

Introduction

The goal of PlayPal was to make a reusable gaming app that has the possibility to swap in and out various games, such as Connect 4 or Checkers. Furthermore, we worked to make a user interface that was user-friendly and intuitive for users with all technological skill levels. Using Redis as the backend storage method allows for countless queries to gather insights about any users or games. While PlayPal is a simple example of using Redis to store a lot of information, this project is buildable, which was another one of our goals. The buildable access is what makes this project especially unique. We could add new games and advance the insights about our users and their statistics.

Methods

Redis Implementation

For our game application, we chose to utilize Redis because of its high-speed in-memory data storage, which would be useful for the game queue. Additionally, Redis is highly scalable and offers several data structures that would be useful, such as lists, hash tables, strings, and sets. We focused on storing a hash table of user data and a hash table of game data, with the intention that these tables link and we are able to gather all the information we would need with Redis queries.

There were multiple programs built to support PlayPal. The main program is *playpal_redis.py* which contains the PlayPI, our Redis-specific API that connects to the Redis database and contains the methods necessary for the generic user-facing *playpal_app.py*, such as inserting users, inserting games, updating skill ratings, obtaining a game history, and more. This file is the only place in all of PlayPal with Redis queries written. Hypothetically, a different API could be developed to store the information that uses a different querying language, such as MySQL. The class of methods from this file is imported into our driver file to run the application.

Generating User and Game Data

To load the initial game and user data, we first had to generate the data using a program called *game_data_generator.py*. This program simulates some number of games played by AI (we

created 100,000 games) playing random moves. The output, *game_data.csv*, captured the game ID (incrementing starting at 0), user IDs of the 2 players, the winner's user ID, the loser's user ID, and the game pattern (a string of the indices of the game pieces played in order). To make the user data, a simple Python script generated a number of users (we made 1000) with incrementing IDs starting at 1 and random alphanumeric passwords.

This user data was inserted into the API as a part of the *load_users_games.py* program which was able to load users at a speed of approximately 69,342 users per second. Every new user registered automatically is set to have a rating of 100 which is a beginner skill level in PlayPal. The skill-level thresholds are as follows.

Beginner: 0 - 150 (rating points never go below 0)
Intermediate: 151 - 350
Advanced: 351 - 500 (rating points never go above 500)

After the users were registered, the games were loaded. For every game loaded in, the API updated the user's rating and skill level. This process allowed us to load 3,483 games per second. As games are won, lost, or tied, users' points increase and decrease accordingly, always taking into consideration the skill level of both opponents. Table 1 illustrates our chosen system of points for games with a winner.

Table 1. PlayPal Points Incrementation System

Player Skill Level	Beginner (L)	Intermediate (L)	Advanced (L)
Beginner (W)	Winner +15 Loser -15	Winner +20 Loser -20	Winner +25 Loser -25
Intermediate (W)	Winner +10 Loser -10	Winner +15 Loser -15	Winner +20 Loser -20
Advanced (W)	Winner +5 Loser -5	Winner +10 Loser -10	Winner +15 Loser -15

After gathering feedback, we determined there should be some points awarded/deducted for tie games. For ties, more points should be awarded to a beginner for tying with an advanced player than an intermediate player. Similarly, an advanced player should not be awarded for tying with a beginner player. Table 2 illustrates the point distribution for tie games.

Table 2. PlayPal Point Distribution for Ties

Player Skill Level	Beginner	Intermediate	Advanced
Beginner	+1 Each	+3 for Beginner -3 for Intermediate	+5 for Beginner -5 for Advanced
Intermediate		+1 Each	+3 for Beginner -3 for Intermediate
Advanced			+1 Each

Tic-Tac-Toe Game and AI Player

For the actual tic-tac-toe game implementation, we created a simple program that allows games to be played directly in the console against an AI player we created. Our first version of the AI player plays any random available move on the board. The table uses row/column coordinates to depict a specific move. When storing the game we transform the coordinates into indices (0 - 8) so that the stored pattern could be a string of numbers. Figure 3 illustrates the game board that our users play on and the board that the game patterns are based on.

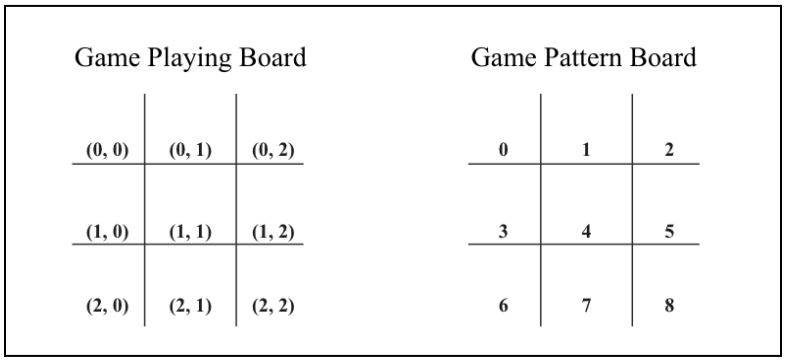

Figure 3. Game Board Translation System

To scale up our AI player, we decided to play around with different levels of reinforcement learning. While testing the learning, both 3x3 boards and 4x4 boards were tested. The final version converged on implementing a basic minimax-inspired algorithm. At this phase, all reinforcement learning is done within *learning_demo.py* and is not connected with the PlayPal app. This is something that could be incorporated into our app for future development.

Game History Table

Additionally, we created a game history table method within the PlayPI class in *playpal_redis.py* that would allow users to see how many games they had won and lost, and their most successful first move that had resulted in a win. This was implemented by retrieving a user's winning game's game history data from the database and running calculations to output their win rate and most common first move played during games that they won. This feature is one option for users when they log into PlayPal and is shown in Table 4 below.

Table 4. Sample Game History Table for Player 4

Wins	Losses	Draws	Win Rate (%)	Most Common First Position
7	8	2	41.1765	(1, 2)

User Interface Implementation

To put all of the pieces together, we made a user interface that was accessible in the console that gives users the opportunity to log in or create a new account. Once logged in, users can play a new game, view game history (using the method mentioned above), or exit the game. If they choose to play a new game, they can choose to play a beginner, intermediate, or advanced player. There is also an option to get paired with an opponent with a random skill level. Then they can actually play the game on the console (against the AI player). After a game, they can play another game, view their game history, or log out.

Throughout the entire creation of the UI, we made sure to leave room for user error, for example, if a user mistypes their password, they are given the opportunity to log in again and when playing the game, all the available moves are listed above the current board.

Reinforcement Learning

In order to train the AI player, we used the minimax algorithm depicted in Figure 5. The tic-tac-toe game is represented as a decision tree with all possible sets of outcomes and points. The program chooses the path that would lead to the maximization of points (winning game) for itself, and the minimization of points for the opponent (losing the game). Every node on the tree is a state of the board based on a move. The tree is built using past game history accessed from the Redis database. The reinforcement learning was added as more of an exploratory arm and has yet to be fully integrated into the main game application, as it currently only plays by itself. Within 470 games played with itself, the AI player was able to learn every possible playable move and consistently win or draw games.

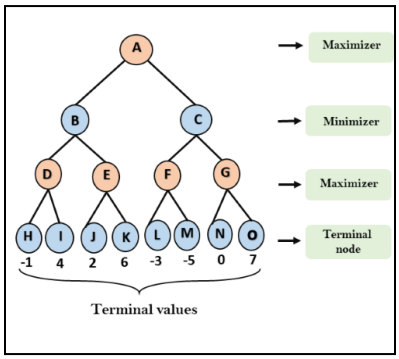

Figure 5. Visualization of Minimax

Analysis

From the 10,000 games played by the random move AI, the most successful first move that resulted in winning games was the middle spot (1,1). However, the best first move with reinforcement learning revealed that a corner move was the best first move. Another insight with using reinforcement learning is that the algorithm is able to solve the 3x3 game in under 500 games, but to solve a 4x4 board (4 in a row wins) takes millions of iterations. In the future it would be interesting to generate a comprehensive visual analysis to demonstrate the reinforcement learning curve in a real-time dashboard.

Conclusions

Our team successfully created a fully functional game application that takes advantage of Redis' high-speed key-value data storage retrieval for the purposes of storing player and game data, matchmaking players, calculating summary statistics, and basic reinforcement learning. The app is able to insert over 69,000 users/per second and over 3,400 games/per second. A human user is able to run a simple app script from their terminal and play with AI in real-time, as well as store and access their data at a later point in time. Through random games, the best first move was the middle piece. Additionally, the architecture of the game application API and files support scalability and reusability. For example, we tested changing the game to a 4x4 board with different rules. Reinforcement learning was also added to the game AI, and the AI was able to learn how to win tic-tac-toe. The main limitation is that we did not get to extend the functionality to support multiple real-time users using the Redis-pub sub or any live chat features. We had initially started with that idea in mind but had pivoted our project after running into issues with threading the server. The UI/UX is relatively basic as well. Currently, a user can only play against the simple AI, as opposed to another human, or the learned AI. Thus, there are still many ways that the project could be expanded upon. However, since the game application is fully functioning and built with reusability and scalability in mind, we are satisfied and would be proud to present this work in the future.

Author Contributions

In the initial stages of the PlayPal project, all group members worked together to come up with a topic and determine all the tasks we would need to complete. Then we broke off and accomplished tasks individually or in groups of two or three.

Joshua Peirce initially created the tic-tac-toe game app and AI player that would eventually be the backend of the tic-tac-toe game in the UI. Josh also worked with Rachel Utama to generate fake game data. Tijana Cosic created the fake user data and also helped to implement the login functionality in the UI. Rachel Cassway and Brady Duncan worked on the PlayPI that held all of the methods supporting PlayPal. Rachel Utama also worked on the game history functionality and Joshua added the game functionality to the UI. Rachel Cassway and Brady did the remainder of the UI implementation by connecting all the pieces that everyone else worked on so we had a fully functioning UI. They also focused a lot on making the UI friendly to user errors to ensure a smooth user experience for all users. Rachel Cassway and Rachel Utama worked on the final report, and Rachel Cassway and Joshua worked on the slide for our in-class presentation.

Overall, the team was able to distribute all the tasks and accomplish everything on time. We worked very well together and were able to lean into each of our strengths to create the best possible project.

References

[1] *Large-Scale Storage and Retrieval*. Homepage - Khoury College of Computer Sciences.(n.d.). Retrieved April 16th, 2023, from https://www.ccs.neu.edu/home/rachlin/nosql/index.html