**GAMBO – DIGITIAL GAMBLING SOFTWEAR**

**Created By Jason Van Humbeck**

Gambo is a sophisticated sampling gambling software developed in Microsoft VBA, leveraging Microsoft Access as its SQL database infrastructure.

**GAMEPLAY** The gameplay is straightforward. Users engage in a simulated dice roll, predicting the outcomes of three dice. Players can place bets, starting at a minimum of $1 CAD, with returns multiplied based on correct predictions

(0 Correct Dice = x0, 1 Correct Dice = x0.5, 2 Correct Dice x3, 3 Correct Dice = x20).

The multiplier increases with each consecutive correct guess. Payouts are calculated by multiplying the multiplier, level of win, and the initial bet. For instance, a sequence of five consecutive wins, a $20 bet, and two correct guesses on the sixth attempt would yield a payout of 5 \* 20 \* 3 = $300.

**DATABASE STRUCTURE** The software utilizes five interconnected databases to manage player information comprehensively:

* Log Database: Tracks user logins and associates them with transaction records.
* Color Database: Stores RGB values for predefined colors, allowing users to personalize their settings.
* Country Database: Contains country data, including name, denomination, and conversion rates, ensuring user familiarity with betting amounts.
* Order Database (Linked to User, and Log): Records all purchases, stores payment details, and links transactions to user and log sessions.
* User Database (Linked with Color and Country): Manages user information such as name, password, UserID, and funds. It incorporates user preferences for color and country.

The databases are live updating through using the application and are consistently read. The databases are read/write in the following order:

* 0) As an automatic load feature the LOG database is called to generate a blank token.
* 1a) The user can login, where it will search through the USER database for existing users which will match the userID and Password entered. If no matches are found it returns an error; if a match is found it follows to step 2).
* 1b) The user can create an account where it will ‘insert’ the entered fields (and automate the rest) into the USER database. It also refers to the COLOR and COUNTRY fields to confirm an entry is valid before allowing it. If any of the above requirements are failed it will return an error message; else it is directed to step)
* 2) Using the above information, this step performs an extra scrape on USER, COLOR, and COUNTRY to ensure all information has been received, and the subsequent screen can get this information easily. It then creates a unique session ID though the LOG database, which will be used for potential orders.
* 3) Before the user can add money or close app, it performs a money “update” through the USER database, where it uses the user conversion rate to change rates back to the uniform Canadian currency.
* 4) If the user chooses to add funds, it will create a unique purchase record through the ORDER database, identifying it through the passed Session, and User IDs.
* 5) After the application has been closed (regardless if it was used), it refers to the LOG database to get the most recent session ID. Using this to then identify the user, and any purchases they made through the ORDER database.

**KEY FEATURES** The application supports user registration, allowing for the creation of accounts seamlessly. Additionally, it features a real-time payment system for new users or those with depleted funds, ensuring instant transactions without the need for system restarts or external support.

Secondly, the software dynamically updates currency based on the user's geographical location.

Lastly, Post-game, the application automatically generates a high-score table highlighting top earners, accompanied by an Excel chart contrasting earnings against expenditures. Furthermore, it produces a detailed transaction receipt summarizing all activities conducted during the login session.

**IMPLEMENTATION DIFFICULTIES:** While working on thisapplication I had to learn how to deal with unique error codes which I was unfamiliar with, this took extensive debugging. Secondly, I had to learn how to use the SQL .update .insert .execute functions, as these are not generally covered in course content. Lastly, I had to learn how to build a database, and how to properly access it.

**HELP** I sought no outside help and designed my code and user database both from scratch. However I did use general forums to understand errors, and use Microsoft’s support engine to find help on new topics, such as SQL and Word generation.

**IMAGES**

A screenshot of a computer program

Description automatically generated

Here is the center of one of my most difficult implementations. You can see it can perform two functions based upon the parameter. It either gets the new live updated money count and saves it, or updates the database based upon its saved count. This required a high learning curve to get this and its parallel functions to work in sync.

A graph with red and green bars

Description automatically generated

Here is an example of the output that it may produce at the end of a game (or app closing), where it not only displays the live leaderboard, but also produces a graph for proper context.

A screenshot of a computer

Description automatically generated

Here you can see the update display function, which ensures that the changing text fields are redrawn, and in sync.