Names: Joshua Licari and Christian Millard

Rock Paper Scissors Ultra Deluxe™

General Overview:

The main function of this project is to play the classic game of “Rock Paper Scissors” with a roguelike twist. We all know and love “Rock Paper Scissors,” but have we ever had those fun times when our younger selves created new hand gestures to be playable in a typical “Rock Paper Scissors” game? Well, that is what this software aims to do (except for running free with your imagination to create the hand gestures yourself).

The problem that this software will be addressing is the fact that this was never visualized or simulated before. The closest thing to this idea is the whatbeatsrock.com website, which uses artificial intelligence to decide what hand gestures (and many more things that you could think of) to win or lose with assumptions. If you want to let your imagination run wild, go play there, but that is what we don’t want to pivot towards. The website isn’t centered on the typical “Rock Paper Scissors” layout rule where the player has free rein to think and pick an option to try and win against their opponent. And for that, we want to expand upon this idea to the next level. Especially the roguelike game loop aspect of this game implemented into the “Rock Paper Scissors” environment.

Software Analysis:

To tackle this problem, we need to lay out some guidelines for this project. For one, it will be a game match between the user and a computer or bot. We will reference this computer or bot as a CPU for the rest of this design document. Two, the user has to have a fair gameplay position no matter how poor their current condition is. There are random elements involved in a typical “Rock Paper Scissors” game, but sometimes the idea is about how the user can get better winning odds through the newly provided hand weapons. Three, information to the user is important. The average user may know the simple rules of “Rock Paper Scissors,” but there is more to this game that needs to be explained and provided so the user can bring their fullest potential in each gameplay experience. Therefore, the objective of this software is to bring a fun and new addictive experience to the user while not being too overly complicated. There has to be a striking balance between these large main issues about what makes a game unenjoyable in the first place.

First, the user will experience the functionality of the software through these game steps (better visualized by the flowchart algorithm at the near end of the document). Here is how it goes:

1. The user will be given an introduction and instructions on the game. After that, the user is prompted to type “start” to begin the game.
    a. The in-game terms for hitpoints and damage will be abbreviated to HP and DMG respectively in the entire game (and this design document).
    b. Instructions include a few small tips that can help the user a better understanding of how to win the games consistently without random luck not being on their side.

Names: Joshua Licari and Christian Millard

2. There are five game battles against the CPU. In each battle, the user and CPU have 10 HP each (except for the first battle, the CPU will have 5 HP because it's basic "Rock Paper Scissors").
    a. The round order goes from the user and then to the CPU. After that, both chosen hand weapons from the user and CPU will happen at the same time.
    b. If the user wins against the CPU's hand weapon, DMG is dealt to the CPU. Likewise happens when the CPU's hand weapon wins against the user's. If both hand weapons tie, no DMG is dealt between the user and CPU.
    c. Once the hand weapon interaction is done, a round is essentially completed (and is kept track within a game).
    d. Each win with a hand weapon will always deal 1 DMG to the opposing side. There are exceptions to certain interactions between opposing hand weapons (refer to the Google Sheet win-loss table in this analysis).
3. After depleting either the user's or CPU's HP, the software decides on the winner and loser. This check happens frequently after a round is completed. A user's win is considered when the CPU's HP depletes to 0 or less. A user's loss is considered when the user's HP depletes to 0 or less.
4. When the user wins the game battle, they're able to choose from three different possible hand weapons (with added description) from a pool of new hand weapons.
    a. Once a new hand weapon is chosen, that hand weapon choice will be removed from the possible pool of hand weapons.
    b. Some new hand weapons are called hand weapon upgrades which remove the base form of a hand weapon already in the user's arsenal and replace it to become upgraded. Plus, the alternative hand weapon upgrades for base forms will be removed.
    c. If less than 3 hand weapons are available, then show only what's left. If all hand weapons are acquired, do nothing.
    d. The CPU also gains a new random hand weapon after every game battle (the CPU cannot acquire hand weapon upgrades).
    e. The CPU gains more HP than the user (5 HP per battle the user won).
5. Repeat steps 2 to 4 until the user wins all five game battles against the CPU or reaches 0 or less HP.
6. The user gets final stats based on their score and revisits what hand weapons they acquired.
    a. The score is added up through each game battle's round wins and subtracted through each game battle's round losses.
    b. The score varies between each hand weapon's win-loss interaction.

Before continuing through the analysis, there is also a larger issue at hand (funny pun). Focusing on the hand weapon interactions aspect of the game, there are too many interactions between many of the new hand weapons (depending on how many are added into the classic "Rock Paper Scissors" game). Considering which beats what, and what beats which, it had to be organized and understandable for both the programmer and user that will be playing through this software. Because of this, we provided a Hand Weapon Win-Loss Table and Descriptions

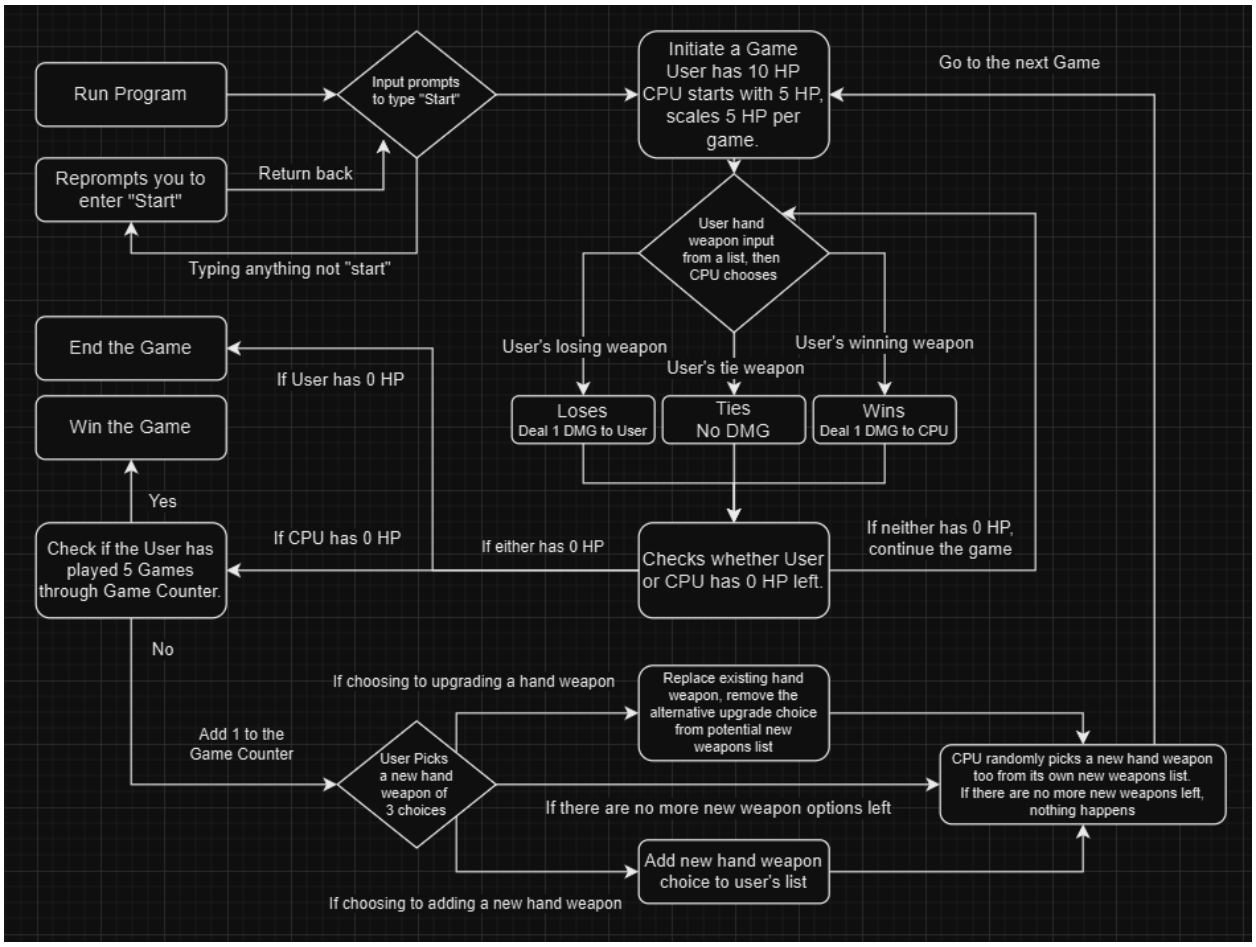
spreadsheet to tackle these problems. Within the Python code, a class is defined to create the "brains" devised with tons of elif statements to determine the winner of a round between two opposing hand weapons. It is neatly sectioned between what matchups would a certain hand weapon do such as a win, lose, tie, or interact differently against.

Continuing, here are some requirements that the software must have to be able to function correctly when programmed to the fullest:

- A declaration of multiple variables and Python lists for the user, CPU, and new hand weapon pool (separate for user and CPU). Plus, more declared variables for game counters, scores, and hand weapon states. This totals up to about 14 or more required variables, and at least 4 variable lists.
- A solid and consistent game loop in the software using the "while" condition until a variable becomes equal or greater than something. This is done to control the amount of games the user can play before ending the software. This also refers to the game counter variable.
- A smooth difficulty scaling and weapon balancing between the user and CPU. As each game battle is won, the next proceeding battle will become harder than before, and each hand weapon interaction to not have any weaknesses (every hand weapon must be strong or weak against something). Provide information and future game knowledge to the user when choosing between the 3 new hand weapons after winning a game battle.
- A random number generator to grab information from certain things in a list of possible new hand weapons, for the CPU to use their hand weapons in each round, and random events with certain hand weapons to activate. For this case, use the "import random" at the start of the software to implement this function.
- Speaking of implementation, also include "import time" and "import playsound." Note that for playsound to be implemented, it goes through this: "from playsound3 import playsound."
- The software can remove chosen hand weapons from the possible choices given to the user (specifically upgrades).
- The software can detect whether there are fewer than 3 possible hand weapons to be picked for the user and when there are no more hand weapons to choose from. This also applies to the CPU's new hand weapon options.
- The user has inputs to use a hand weapon, and input to choose a hand weapon with numbers. All invalid inputs are addressed back to the user as feedback and they are prompted to type appropriately or within the given options.
- Installing several mp3 audio files as sound effects to bring the user a fun experience with the software. Due to the functional nature of playsound, which is playing an audio file before proceeding on with the next line of code, no background music can be played. This is optional, but it is recommended for the user experience while playing with the software (and is better than anticipated silence).
- Adding in a final score variable and printing a list of used hand weapons at the end of winning or losing the entire game between each hand weapon interaction gives the user the sense of higher goal-reaching opportunities. Since the score is deducted every time

the user loses a round against the CPU, users would want to try to win as consistently as possible.

Flowchart Algorithm:



We discovered this algorithm through the top-down strategy, where the algorithm has a procedure for making decisions from the top when certain events happen to the bottom decisive victor as the multiple game battles become expansive over time. In this case for the software, the iterative algorithm has its procedure when picking the winner between the two opposing hand weapons chosen by the user and CPU, having a game battle loop until the user has completed five game battles, and deciding whether the hand weapon's ability effects work through chance. There are more as seen on the flowchart, but the more important deciding factors are these three situations. While the bottom-up strategy sounds like how this software's algorithm was developed through the building blocks to the ultimate decider, the top-down strategy creates this hierarchy to take into account all possible hand weapon win-loss interactions that the bottom-up strategy couldn't do.