I've been meaning to make a game for a while now, and I think I know just what I'll do.

**Idea:**

Gameplay would revolve around an on-screen calculator. Two players would be shown the same random number and be tasked with performing a series of math operations to reach that goal, only being able to input the numbers 0-9. Whenever the goal number is reached, that player gets a point, and a new random number is selected. Play continues until one player reaches a set amount of points or until a timer expires, with whoever achieves the higher score winning.

There would also be a single player "time trial" mode, where players would be fed a given quantity of numbers and timed to see how long it takes to reach them under the same constraints as the multiplayer mode. That is, this would be implemented if I have the time to do so.

There would also be a standard calculator included as well.

**Features:**

* Calculator with the following capabilities:

1. 0-9 number pad with decimal place
2. All basic arithmetic operators (addition, subtraction, multiplication, and division), as well as modulus
3. Parenthesis and grouping
4. Square root buttons and Exponents (X^Y), as well as a simple square button (X^2)
5. Trigonomic functions (sine, cosine, and tangent)

* Main menu with buttons for Single player, Multiplayer, Calculator Mode, Options Menu(explained later), Help, and quit.
* Sound effects as well as music. (potentially)
* Options menu allowing to edit/alter:

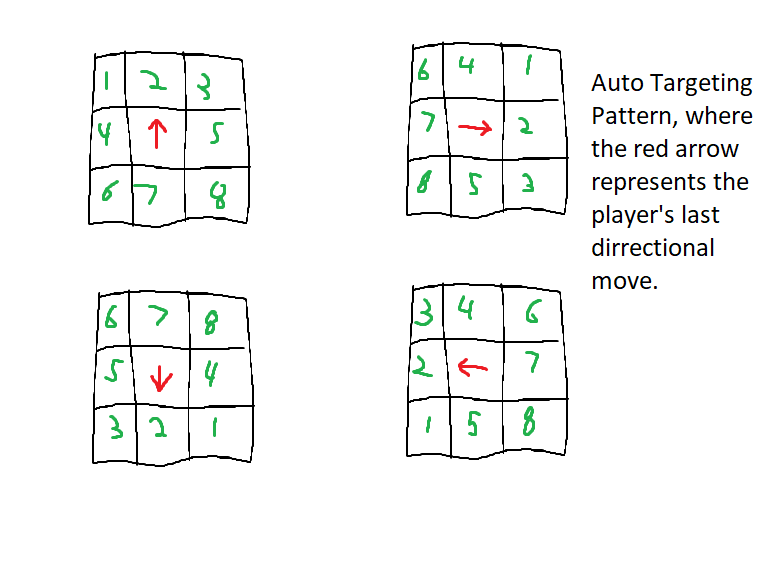
1. Sound effects and Music Volume
2. Rebinding of controls
3. A list of gameplay modifiers and an ability to create and save new modification presets (explained later)

* Pause Menu with options to resume game, change certain (non gameplay) settings, and an option to quit to the main menu.
* Up to 4 player multiplayer, though this would be intended for those with multiple keyboards (potentially)

**Combat:**

Because player cursors cannot overlap, one could create a situation where one player obtains a lead in points and then cursors over to the “Submit” button, meaning other players cannot complete their equations and thus allowing the one in the lead to simply wait out the clock. Therefore, rather than take the simple approach of making the “submit equation” button a physical button on a controller/keyboard, I instead intend to implement a combat system. Upon pressing the “combat” button, the player's cursor would “punch” an opponent's cursor if said opponent was within 1 tile of the combatants position. Detection is done by scanning the area surrounding the attacker, starting at the This “punch” would auto target to the opponent in the players most recent of movement, as seen in Fig. 1, however the button could be held, with any movement input being interpreted as a targeting direction (you move left, you punch left, you move up, you punch up). Upon being hit, a player would be sent backwards a tile (unless they are on a border, in which case nothing happens), be put in a "stunned" state where they cannot move, punch, or take damage for 1 second, and would receive damage. After being hit a certain amount of times, that player would die, and would have to wait for a timer to expire before they can respawn. 1 point of health is recovered for every two seconds the player is not in combat (not punch and not being punched), and all player health would reset once a point is scored.

**Fig 1**



**Gameplay "Modifiers:"**

Put simply, these would be a list of settings allowing users to alter certain gameplay values and internal variables that can radically alter the game state, similar in concept to how Nintendo’s landmark *Super Smash Bros.* series allows for custom rulesets. Furthermore, these settings would be saved separately from the sound/control settings, meaning users could create "presets" of configurations they like, and pick which preset they wish to use on the fly. Included in these presets would be a series of presets I would have created. What exactly these presets would be is still undetermined, but some ideas include:

* A mode which removes everything except for the basic 0-9 positive integer arithmetic, meaning only addition, subtraction, multiplication, and whole number division is allowed (1/2 would return ERR, while 4/2 would return 2)
* A mode which removes combat from the game entirely, allowing for players to phase through each other.
* A mode where health is not regenerated passively, with a “health pack” being placed on the board every 30 seconds, with whoever grabs it first receiving a complete heal to full health.
* A mode which removes PEMDAS, i.e. 1+3\*2 would be 8, because 1+3=4, and 4\*2=8, instead of 7. The only exception to this is parenthesis, which are still calculated before everything else.
* A mode which removes complex equations, i.e. 5+7-2 isn’t allowed, with players instead needing to input 5+7, then ANS-2.

**Implementation and Expected Challenges:**

There is a rather large list of potential issues, though some are much smaller than the rest. In fact, there’s only really one potential challenge that might hinder the prospect of creating this game. I will, however, go over as many potential issues I can think of, no matter how minor they are.

The first issue is that of language. Put simply, I’m not to familiar with Java or Javascript, I’m far to out of practice with C# to create anything useful in the short time frame, and I personally find C++ about as human friendly as writing raw hex bytes, so I’m basically limited to just Python, which I am very much familiar with. This, however, raises the question of performance. Though Python has several libraries that could in theory handle a game of this scope, such as pygame, due to Python being an interpreted language, execution is poorly optimized, meaning I am limited in what I can do with the tools I have. This, however, should pose no real problem. I am well-versed in Python and pygame and their limitations, and have designed this project with that in mind. The simple nature of this game being based around a calculator means that most of the graphics are just squares with text written on them, meaning that drawing the graphics should be rather easy for a modern computer. Furthermore, thanks to complete control of the drawing routine, I can decide exactly where and when any given element is drawn, and how it’s created.

Another is of scope. Such a lofty project might not be able to be completed in five week’s time, right? While that may be true for someone inexperienced with Python and programming as a whole, I, again, have years of experience in Python, and know my programming skills. In fact, I had actually attempted at one point to create a game with a similar concept years ago, though the files have long since been deleted. I am completely confident that I can write the core gameplay code within a week, the multiplayer stuff in four days, the Singleplayer in two, and the entire menu system in maybe five, leaving me three weeks to add new features, test for bugs, and polish the look of the game to remove the simple boxes with text on them and replace them with actual buttons. Heck, I might even find the time to implement gamepad support, complete with button rebinding for each individual player.

This brings me to the actual issue that plagues this project: the core gameplay code, namely the calculator itself. While programming a simple calculator with only the four key arithmatic operations and maybe a decimal place, this is something far greater than just that. Because of the wide number of operations that the calculator can perform, as well as the inclusion of parenthesis, calculation simply cannot be done in the same way it is for most calculator programs. Instead, I must create an algorithm that takes a string of text, parses it into a mathematical equation, determine if said equation is legitimate (no division by zero, all the numbers are in place, etc.), and then calculates said equation. In short, I can’t simply tell a computer to add 5 and 7, I need it to understand what “5+7” means. This can be done, as programs and even higher end calculators have been built before with this functionality, however something like this is much farther out of my comfort zone as a programmer than something like adding .wav files to an animation would be. Furthermore, those people weren’t doing it in an interpreted language, in the context of this being a core action in a video game, meaning it has to be done practically instantly. However the good news is that once I finish this part of the game, the rest should be easy enough to design and code.