1. Design character and opponent animation frames
   1. I found a method that seems to work well but is tedious so a better solution would be +
      1. Look up “karate kick” or “karate punch silhouette” on google images
      2. Pick a stance to model the animation frame after
      3. Use a pixel art program, such as “PixilArt” which runs in browser, to draw a custom version of that stance.
      4. Make sure the pixel art version you create is 48 pixels high by 32 pixels wide.
      5. Open a text file and map the pixel art version to binary code ie;
         1. 00000000 00001110 00011000 00000000
         2. 00000000 00001111 00001100 00000000
         3. ……………………………………………………………….
         4. Note that there should be 48 rows and 4 columns (48 bits x 32 bits)
      6. Use the helper function in our github called “drawBinary” to print your current binary file to screen with all 1s replaced with ‘\*’ and all 0s replaced with a space to get a better visual representation since 1s and 0s can become unintelligible.
         1. It takes a command line argument which would be the text file you’re creating.
      7. Use the helper function in our github called “binaryToHexColumnwise” to convert this file of bits to hexadecimal notation for use in the assembly program.
         1. It takes a command line argument which would be the text file you created.
2. Design intro screen art
   1. Street Fighter 5 custom text
   2. Strips of color above and below the text that flash or have some effect?
3. Design a start menu/intro screen menu
   1. Menu items
      1. Start game
      2. Quit
      3. Selector Icon
         1. To indicate which option they are currently on
      4. Or alternatively, we don’t use a menu and just print “press [some key] to start”
4. Develop an organized RAM configuration that we will use for the implementation
   1. ram\_00 character x position
   2. ram\_01 character y position
   3. ram\_02 opponent x position
   4. ram\_03 ….
   5. Etc.
   6. This will serve as a set of rules that determine what RAM we use for implementation to make sure we don’t get disorganized and overwrite RAM that is storing something important
   7. It is probably a good idea to keep related information grouped together in “chunks” with two or more unused RAM locations after each chunk in case we forget something that we want to add in later.
      1. It also isn’t a great idea to have too much unused RAM since memory usage is a marking criteria so it would be even better if we could brainstorm super hard on this to make as concise a mapping as we can for all variables we will need then just use RAM after this if we forget something
5. Figure out how to generate random numbers (for opponent AI)
6. Develop opponent AI
   1. Use random number generation to determine when the opponent kicks, punches, moves, and how often he blocks incoming strikes.
   2. If he is being struck, generate a number between 0 and 100 and if it’s greater than 60, block – alter for difficulty levels
   3. If he isn’t being struck, generate a number similarly that determines if he moves or strikes
7. Design the HUD
   1. Character and opponent health bars
   2. Timer?
   3. Score
   4. These are important variables to map to RAM in point 4.

It would probably be a good idea to have a punch and a kick animation for the opponent, develop some very basic opponent AI, develop a preliminary HUD, and map the RAM locations as precisely and thoroughly as possible for the first project update and worry about the start screen, additional animations, AI tweaking, a more advanced HUD and stuff like that for the final product.