Module Descriptions

10/14/2016

# Game.py

Holds all data for all sports including team and player data.

Game class is base class and each sport has a subclass. Only redefined functions are in the subclasses.

Every key press event will have its own named function to properly manipulate the data.

Any key press event that does not require immediate data change will do nothing and be handled further by the Menu\_Event\_Handler class.

Game module methods reach in to Team and Player for their data.

Uses the userConfig file, gameUserSettings file, segmentTimerUserSettings file, and gameDefaultValues.csv.

# Team.py

Holds all data for all teams including player data.

Uses teamDefaultValues.csv file.

# Player.py

Holds all data for all players.

Uses playerDefaultValues.csv file.

# Config.py

Creates the default and user config files. Contains methods to modify the user files.

# GameDefaultSettings.py

Creates the default and user gameSettings files. Contains methods to modify the user files.

# SegmentTimerDefaultSettings.py

Creates the default and user segmentTimerSettings files. Contains methods to modify the user files.

# functions.py

Holds all global functions.

# clock.py

Scoreboard clock timers class for all types.

# Console.py

Simulates a console only. Consists of these objects:

16x2 LCD menu output, keypad, game data, mp protocol encoder/decoder, mp address maps, config file, checkEventsTimer (simulated mp interrupt only faster), updateMPDataTimer (mp data sent every tenth of a second)

# Keypad\_Mapping.py

Simulates a keypad by mapping a key press to the selected game method. All combinations of x=B to F, y=8 to 1 are in the grid. Physical correlation of this grid to the keypad hardware is needed to map.

Uses MP\_Keypad\_Layouts.csv file.

# Menu\_Class.py

Simulates the 16 x 2 LCD screen. It is a complicated state machine. It starts in a non-menu state and waits for a key press it cares about then reacts based on that menu function name. It remembers previous states and follows the calculated path to either exiting the menu or waits for the external menu timer to reset it.

The Map method is the root method called for a key press event. It receives a functionString and the game data then saves the functionString, calls the functionString named method, calls the UpdateMenu method, and checks to see if we have exited and switches to the current sports default screen. Finally, it returns the game data with any changes.

The main update function which branches to a sub-method based on key pressed and previous state.

\*\*Sub-methods\*\*

\* Start function = First call to this menu from the default screen.

\* Self function = Same key pressed as the current menu.

\* Exit function = Key pressed has a special menu close function.

\* Clear function = Clear key is pressed.

\* Enter function = Enter key is pressed.

\* Number pad function = Keys 0 through 9 is pressed.

\* Do nothing function = Key press did not trigger any other sub-functions.

Uses MenuMap.csv file.

# MP\_Data\_Handler.py

This object is just used to hold the methods that encode and decode the mp architecture protocols. No data is stored for use in this object.

# Address\_Mapping.py

This object deals with the enormous cluster headache of the mp code starting with the concept of an internal value of named data and perverting it into a scatter plot of segments all with their own individual rules that if physically positioned together, with the right trimmings, can trick the brain into thinking it sees a scoreboard.

### Initialization

Build a dictionary of the 32 possible words of a sport named \*\*wordDict\*\*. This will be the default values for that sport.

Key = binary address of group and bank

Value = a 16-bit word in the low-byte then high-byte format

Word in this context is one packet of information a driver needs to update a single header or headers.

=========== ===========

High Byte Low Byte

=========== ===========

1GBBWWIH 0GFEDCBA

=========== ===========

G = Group, B = Bank, W = Word (Different than the word mentioned above)

I = Control bit, H through A = The eight segments of display data

Note: Low byte is received first and the high byte is received second

The Map method updates the list of MP Format data packs to be sent after building the current map with any alts changed.

The adjustAllBanks chain

adjustAllBanks first calls \_updateAddrWords

\_updateAddrWords calls \_loadFromAddDict

\_loadFromAddDict calls mp.Encode

Current Dict of 32 words are up to date

adjustAllBanks next calls \_updateSendList

Only load values that have changed in to the send list

Note: Map and UnMap are the only publically callable methods for this class.

UnMap is for doing the reverse and interpreting mp protocol words but uses the same csv file.

Uses AddressMap.csv file.

# Scoreboard.py

## Subclass of Console – Inherits all of Console then adds or modifies methods

Simulates a scoreboard with a built-in console.

# UI\_Scoreboard\_Parts.py

This collection of classes contains all of the graphical objects we use to draw a scoreboard. The Board class uses all of the other classes.

Uses the LED\_Positions.csv, Masks\_Parts.csv, Masks\_Per\_Model.csv, and Chassis\_Parts.csv files.

# UI\_Simulator.py

Application to select and simulate a console and scoreboard.

Uses the Models\_CX\_Jumper\_Defaults.csv file.

# Driver.py

Simulates a driver.

Uses the file.