Perfect Game Baseball - msherwood pseudo code backbone
Using “MVC” Convention (Model/View/Controller) With “Multimodal I/O”

**Model:**
DataModel.h/c
  Use this to store the game states

**View:**
Buzzers.h/.c
  Output sound effects (supplanted by Speaker)
Shiftregister.h/c
  Output visual effects- based off the model, so once model is updated just call viewUpdate
Speaker.h/c
  Output sound effects

**Controller:**
ADS12.h/c
  Give time difference control
Delay.h/c
  Calculate delay periods
Main.h/c
  Main controller function- run everything
PWMS.h/c
  Control pulse width modulation and variations

**Input:**
Multiplexer.h/c
  Get input from hit balls, user pitch control, automatic pitch sensing, and swag release
Penny.h/c
  Break beam sensor control for penny input

**Output:**
Motors.h/c
  Run pitch motors and ball elevator (not sensing)
Swag.h/c
  Run swag release (not sensing)

**MAIN Controller**
{
  Run Startup Sequence (call all of the modules' init functions and init timer/PWMS)
  Update Scoreboard in View
  Play Anthem in View
  Loop forever
    Wait for a penny in Input
    Set Game To On in the Data Model
    Update Scoreboard in View
    Play sound at start in View
    Loop, while game is not over
      Wait for pitch in Input
      Deliver Pitch in Controller using (Output and Input)
      Wait for pitched ball to register as an outcome in Input
      Log outcome as event type in Controller
      Handle the outcome of the hist change Model and update View
/**
 * Function: static void Run_Startup(void)
 * Initializes all the C32 output pins through
 * the modules' init functions. Also initializes the
 * ** TIMER.
 */

Startup sequence Controller
{
    Initialize Timers
    Initialize Coin Collector
    Initialize Multiplexer
    Initialize Data Model
    Initialize Shift Register
    Initialize A/D
    Initialize Motors
    Initialize Speakers
    Initialize Swag
}

/***
 * Function: static void Handle_Hit(char event)
 * Handles the logic of a hit event. Access the data model
 * and updates the view/outputs
 */

Handle Hit Controller
{
    case HOME_RUN:
        Tally Baserunners and add 1 in Model
        Play sound in View
        Blink Scoreboard in View
        Add runs for baserunners plus 1 in Model
        Clear Bases in Model
        Clear Strikes in Model
        break
    case TRIPLE:
        Play sound in View
        Count baserunners in Controller logic
        Add runs for baserunners in Model
        Put runner on third in Model
        Clear Strikes in Model
        break
    case DOUBLE:
        Play sound in View
        Count baserunners in Controller logic
        Add runs for baserunners in Model
        Put man on second in Model
        Clear strikes in Model
        break
    case SINGLE:
        Play sound in View
        Tally baserunners in Controller
        Score runs in Controller
        Put runner(s) on base in Model
        Clear Strikes in Model
case STRIKE:
    Check for strikeout in Controller
    If strikeout play sound, blink scoreboard, add out in View and Model
    If not strikeout, add strike in Model

case OUT:
    Clear strikes in Model
    Add Out; in Model
    break;

Update scoreboard in View
Check end of game in Controller
}
Start Stepper Motor in Output
Wait for Break Beam in Input
Stop stepper motor in Output
Continue to run pitching motors until ball is delivered in Input and Output

*****
** Function: void RunPitchingMotors(char input)
** If input is high, turns both pitching motors on to a duty cycle between
** MIN_PITCH_PWM and MAX_PITCH_PWM.
** dutyCycle = MIN_PITCH_PWM + (0-100)*(Max_PWM - Min_PWM)/100
*****
RunPitchingMotors(char input) Output
{
    Check speed and curve inputs
    Set pulse widths per algorithms
    Turn motors on
}

*****
** Function: char CheckHit(void)
** Loops on the multiplexer to find a switch closed event.
** Use pins T5,6,7 for multiplexer control, T4 for mux out.
** @returns a symbol event of what happened
*****
Check Hit Input
{
    Cycle through MUX channels 0-7. Start with 0...
    Loop, forever
        Timeout if needed
    Return outcome (single, double, etc)
}

*****
** Function: char CheckPitch(void)
** Loops on the multiplexer to find a switch closed event (for the pitcher).
** Use pins T5,6,7 for multiplexer control, T4 for mux out.
** @returns a symbol of pitch happening
*****
Check Pitch Input
{
    Wait for pitch button to be pushed
    Return Pitch
}

*****
** Function: void CollectPenny(void)
** Waits for a penny insertion.
*****
CollectPenny Input
{
    Wait for penny to be deposited by user
}

*****
** Function: void UpdateShifter(void)
** Updates the LEDs 0-15. Every time.
*****
UpdateShifter(void) View
{
    Feed in values for scoreboard serially pulsing clock between each bit
}

/*****
** Function: void SwagIO(char winner)
** Releases swag to the winner, passed as a symbol constant.
*****
SwagIO Output
{
    If batter won
        Deliver swag to batter
    If pitcher won
        Deliver swag to pitcher
}