/Users/leagueduser/Desktop/ME_218C_Project/ProjectSource/GameplayFSM.c

```
****************************************************************************
Module
  GameplayFSM.c

Revision
  1.0.1

Description
  This module handles the states of the game while the subnetmes communicate with one another and try to sink others' ships!

Notes

History
When           Who     What/Why
-------------- ---     --------
03/24/20       Lyl     started writing our gameplay framework
01/15/12 11:12 pm      revisions for Gen2 framework
11/07/11 11:26 pm      made the queue static
10/20/11 17:55 pm      fixed references to CurrentEvent in RunTemplateSM()
10/24/11 18:20 pm      began conversion from SM Template c (02/20/07 rev)
****************************************************************************/
/*----------------------------- Include Files -----------------------------*/
/* include header files for this state machine as well as any machines at the
   next lower level in the hierarchy that are sub-machines to this machine
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "GameplayFSM.h"
#include "ThisPartyService.h"
#include "ServoService.h"
#include "TXService.h"
#include "TX_Service.h"
#include "GenerateStationMessages.h"
#include "dpsim0.h"

/*----------------------------- Module Defines ----------------------------*/
#define FULL_SLOT_TIME 200
#define QUIET_TIME 10
#define BIT_TIME 20
#define SERVO_DELAY 40
/*----------------------------- Module Function ---------------------------*/
static void CheckSyncAndTransition(void);
/*----------------------------- Module Types      -------------------------*/
static GameState_t CurrentState;
static CurrentGameStatus_t CurrentStatus;
static bool PlayingSMNotDoneFlag = false;
static uint8_t MyPriority;
static GameStatus_t GameStatus;
static uint16_t TIMA_FRAME_TIME;
static bool HeardFirstBit = false;
static bool TestBurnerEnabled = true;

/*----------------------------- Module Code       -------------------------*/
****************************************************************************
Function
  InitGameplayFSM

Parameters
  uint8_t : the priority of this service

Returns
  bool, false if error in initialization, true otherwise

Description
  sets every the priority, sets up the initial transition and does any
  other required initialization for this state machine
Notes

Author
  Lisa Gardner, 5/24/20
****************************************************************************
bool InitGameplayFSM(uint8_t Priority)
{
  ES_Event_t ThisEvent;

  MyPriority = Priority;
  gameStatus = getPointerToGameStatus();// get the pointer to update the game status
  gameStatus->SetHeading = TP;// set the initial heading
  gameStatus->SetPosition[X_=7];// set up the x and y position for the initial sub position
  gameStatus->SetPosition[Y_=7];
  InitializeGameStatus();// it's the strict static values to start the game
  TIMA_FRAME_TIME = gameStatus->NumSlots * (FULL_SLOT_TIME);// set the entire TDMA frame time based on number of teams
  CurrentState = WaitingToPlay;// set the initial state of the game to WaitingToPlay
  CurrentStatus = Sync;// set the stress of game is Syncing phase
  ThisEvent.EventType = ES_INIT;// post the initialization event
  if (ES_PostToService(MyPriority, ThisEvent) == true)
  {
    return true;
  }
  else
  {
    return false;
  }
}

****************************************************************************
Function
  PostGameplayFSM

Parameters
  EF_Event_t ThisEvent , the event to post to the queue

Returns
  boolean False if the Enqueue operation failed, True otherwise

Description
   Posts an event to this state machine's queue
Notes

Author
  Lisa Gardner, 5/24/20
****************************************************************************
bool PostGameplayFSM(ES_Event_t ThisEvent)
{
return ES_PostToService(MyPriority, ThisEvent);
}

****************************************************************************
Function
  RunGameplayFSM

Parameters
  ES_ThisEvent_t : the event to process

Returns
  ES_Event_t, ES_NO_EVENT if no error ES_ERROR otherwise

Description
  Handles receiving and transmitting during the game
Notes

Author
  Daniel Chan & Lisa Gardner 5/24/20
****************************************************************************
ES_Event_t RunGameplayFSM(ES_Event_t ThisEvent)
{
  ES_Event_t ReturnEvent;
  ReturnEvent.EventType = ES_NO_EVENT;// assume no errors
  ES_Event_t NewEvent;

  switch (CurrentState)
  {
  case WaitingToPlay:       // If current state is WaitingToPlay
    {
    if (ThisEvent.EventType == ES_START_SYNC)   // if the current event is ES_START_SYNC
    {
      ES_Timer_InitTimer(TIMA_FRAME, TIMA_FRAME_TIME - BIT_TIME);// start a full frame timer
      NewEvent.EventType = ES_START_LISTENING;// post start listenning event to receiving
      PostRXServiceNewEvent();
      CurrentState = Receiving;// set the CurrentState to Receiving
    }
    }
    break;

  case Receiving:
    {
    if((gameStatus->TheyAreAlive == 0) && (gameStatus->WeAreAlive != 0)) || ((gameStatus->TheyAreAlive == (0BTFF0B)) || (gameStatus->TheyAreAlive == (0BTF11B)) || (gameStatus->TheyAreAlive == (0BTF2B)) || (gameStatus->TheyAreAlive == (0BTF30B)) || (gameStatus->TheyAreAlive == (0BTF40B)) || (gameStatus->TheyAreAlive == (0BTF50B)) || (gameStatus->TheyAreAlive == (0BTF60B)) || (gameStatus->TheyAreAlive == (0BTF70B)) && (gameStatus->WeAreAlive == 0)) {  // if only one team is alive
      ES_Event_t NewEvent;
      NewEvent.EventType = ES_GAMEOVER;// post ES_GAMEOVER event to GameplayFSM
      PostGameplayFSMNewEvent();
    }
    switch (ThisEvent.EventType)
    {
      case ES_TIMEOUT:
      {
        ES_Timer_InitTimer(TIMA_CLOCK_FULL_SLOT_TIME);// always reinitialize clock

        if (ThisEvent.EventParam == TIMA_FRAME) {// if we are the first ones here (we have not heard anyone else for a whole frame)
        NewEvent.EventType = ES_STOP_LISTENING;// post ES_STOP_LISTENING to RXService
        PostRXServiceNewEvent();
        ES_Timer_InitTimer(SERVO_DELAY_TIMER, SERVO_DELAY);// start the SERVO_DELAY_TIMER to allow time for the servo to PTT before transmitting
        ES_Event_t NewEvent;
        NewEvent.EventType = ES_PRESS_KEYBOARD;// post ES_PRESS_KEYBOARD to ServoService
        PostServoServiceNewEvent();
      }

        if(ThisEvent.EventParam == SERVO_DELAY_TIMER) { // if the servo delay timer expires
          if (HeardFirstBit) { // if we have heard the first bit
            gameStatus->ExpectedNextSlotNumber += 1;// increment the ExpectedNextSlotNumber
            if
            if(gameStatus->ExpectedNextSlotNumber == (gameStatus->OurTeamNum) * 3 ) { // if the expected next slot number is CONN's
              NewEvent.EventParam = CONN;// set the event parameter to CONN
            }
            else if (gameStatus->ExpectedNextSlotNumber == (gameStatus->OurTeamNum) * 3 + 1) { // if the expected next slot number is SONAR's
              NewEvent.EventParam = SONAR;// set the event parameter to SONAR
            }
            else if (gameStatus->ExpectedNextSlotNumber == (gameStatus->OurTeamNum) * 3 + 2) { // if the expected next slot number is TORPEDO's
              NewEvent.EventParam = TORPEDO;// set the event parameter to TORPEDO
            }
            NewEvent.EventType = ES_START_TRANSMIT;// Post ES_START_TRANSMIT event to TXService
            PostTXServiceNewEvent();
            CurrentState = Transmitting;// Set CurrentState to Transmitting
            HeardFirstBit = true;// Set HeardFirstBit to true
          }

          else if(ThisEvent.EventParam == TIMA_CLOCK) {
            CheckSyncAndTransition();// check to see if we should make a transition
            printf("%%%%%%SLOT %d, NUM MOVES ON: %d %%", (gameStatus->ExpectedNextSlotNumber), gameStatus->NumMovesOnIsAlive);

            // if it's our turn
            if (gameStatus->ConnActive) && (gameStatus->ExpectedNextSlotNumber == (gameStatus->OurTeamNum) * 3 ))
            || (gameStatus->SonarActive) && (gameStatus->ExpectedNextSlotNumber == (gameStatus->OurTeamNum) * 3 + 1)
            || (gameStatus->TorpedoActive) && (gameStatus->ExpectedNextSlotNumber == (gameStatus->OurTeamNum) * 3 + 2))
            {
              NewEvent.EventType = ES_STOP_LISTENING;// post ES_STOP_LISTENING to RXService
              PostRXServiceNewEvent();
              ES_Timer_InitTimer(SERVO_DELAY_TIMER, SERVO_DELAY);// start SERVO_DELAY_TIMER
              ES_Event_t NewEvent;
              NewEvent.EventType = ES_PRESS_KEYBOARD;// push the servo down to start PTT
              PostServoServiceNewEvent();
              printf("%OUR TURN    %");
            }
          }
          gameStatus->ExpectedNextSlotNumber += 1;// increase slot number
        }
        break;
        case ES_IOR:
        {
          if(HeardFirstBit) { // if we haven't heard anything before this
            ES_Timer_InitTimer(TIMA_CLOCK_QUIET_TIME);// set a timer to expire for the remaining time in the slot
            ES_Timer_StopTimer(TIMA_FRAME);// stop the timer of the entire
            HeardFirstBit = true;// now we have heard the first bit
          }
        }
        break;

        case ES_BAD_RECEPTION:
        {
          if(CurrentStatus == Sync) {
            ES_Timer_StopTimer(TIMA_FRAME);
            ES_Timer_InitTimer(TIMA_CLOCK_QUIET_TIME);
            HeardFirstBit = true;
          }
        }
        break;

        case ES_GAMEOVER: {
          NewEvent.EventType = ES_STOP_LISTENING;
          ES_Timer_StopTimer(TIMA_CLOCK);
          PostRXServiceNewEvent();
          printf("%GAMEOVER!!!%");
        }
        break;
        } // end of switch case
      } // end of receiving
    break;

    case Transmitting:       // If current state is state one
    {
    if((gameStatus->TheyAreAlive == 0) && (gameStatus->WeAreAlive)) || ((gameStatus->TheyAreAlive == (0BTFF0B)) || (gameStatus->TheyAreAlive == (0BTF11B)) || (gameStatus->TheyAreAlive == (0BTF2B)) || (gameStatus->TheyAreAlive == (0BTF30B)) || (gameStatus->TheyAreAlive == (0BTF40B)) || (gameStatus->TheyAreAlive == (0BTF50B)) || (gameStatus->TheyAreAlive == (0BTF60B)) || (gameStatus->TheyAreAlive == (0BTF70B)) && (gameStatus->WeAreAlive == 0)) {
      ES_Event_t NewEvent;
      NewEvent.EventType = ES_GAMEOVER;
      PostGameplayFSMNewEvent();
    }
    switch (ThisEvent.EventType)
    {
      case ES_IOT: // we have finished transmitting
      {  // to me end of the active team
        if (((gameStatus->ConnActive) && (gameStatus->ExpectedNextSlotNumber == (gameStatus->OurTeamNum) * 3))
        || (gameStatus->SonarActive) && (gameStatus->ExpectedNextSlotNumber == (gameStatus->OurTeamNum) * 3 + 1)
        || (gameStatus->TorpedoActive) && (gameStatus->ExpectedNextSlotNumber == (gameStatus->OurTeamNum) * 3 + 2))
        {
          // keep key pressed
        }

        //else our team is done for the round
        else {
          NewEvent.EventType = ES_RELEASE_KEYBOARD;// release PTT
          PostServoServiceNewEvent();// post to ServoService
          CurrentState = Receiving;// set CurrentState to Receiving
        }
      }
      break;

      case ES_TIMEOUT:  // if there's a timeout event
      {
        if (ThisEvent.EventParam == TIMA_CLOCK) { // if the TIMA_CLOCK times out
          printf("%TIMEOUT!!!%");
          ES_Timer_InitTimer(TIMA_CLOCK_FULL_SLOT_TIME);// restart the clock
          gameStatus->ExpectedNextSlotNumber += 1;// increment the expected slot number
          CurrentState = Receiving;// set CurrentState to Receiving
        }

      }
      break;

      case ES_GAMEOVER:
      {
        printf("%GAMEOVER!!!%");
        ES_Timer_StopTimer(TIMA_CLOCK);// stop the clock
      }
      break;
      default:

    }  // end switch on ThisEvent
    } // end of transmitting
    break;
    default:
  }                                   // end switch on Current State
  return ReturnEvent;
}

****************************************************************************
Function
  QueryGameplayFSM

Parameters
  None

Returns
  CurrentGameStatus_t CurrentStatus (Sync, Play, Dead)

Description
  returns the current game status
Notes

Author
  Lisa Gardner, 05/24/2020
****************************************************************************
CurrentGameStatus_t QueryCurrentGameStatus(void)
{
return CurrentStatus;
}

****************************************************************************
Function
  QueryGameplayFSM

Parameters
  None

Returns
  GameState_t The current state of the Gameplay SM

Description
  returns either Sync, Dead, or Play statuses
Notes

Author
  Lisa Gardner, 05/24/2020
****************************************************************************
GameState_t QueryGameplayFSM(void)
{
return CurrentState;
}

****************************************************************************
Function
  CheckSyncAndTransition

Parameters
  None

Returns
  Nothing

Description
  checks the checking of the round's status and when a transition should be made. Also handles queuing of the desired action for the round.
Notes

Author
  Holly Francis, 05/25/2020
****************************************************************************
static void CheckSyncAndTransition(void){
  if (gameStatus->ExpectedNextSlotNumber == (gameStatus->NumSlots)) { // if we've reached the end of a round
    if (gameStatus->NumMovesOnIsAlive == (gameStatus->NumSlots)) {
      if (CurrentStatus == Dead) {// (gameStatus->isItMyOnAlreadyon == true) { // if we have received enough SYNCS/NEXT_TURNS/NO_ACTIONS or have manually overrode the game flow
        if(CurrentStatus == Sync) { // if we are in sync move on to dead time
          CurrentStatus = Dead;
          gameStatus->isItMyOnAlreadyon = false;
          printf("%*** DEAD!!!------------------------------------------DEAD!!!***%");
        }
        else if (CurrentStatus == Play ) {
          CurrentStatus = Dead;// we are in play move on to dead time and increment the turn number
          gameStatus->Turn += 1;
          printf("%*** DEAD! (TURN %d)------------------------------***%", gameStatus->Turn);
        printGameStatus();//This is true for a full protocol, false to just communicate with gui
        resetGameStatus();//This should occur after every turn, just resets info on our and enemies' moves during each turn
        }
        else {
          CurrentStatus = Play;
          printf("%*** PLAYING! (TURN %d)------------------------------***%", gameStatus->Turn);
          if(gameStatus->ConnActive) { // if Conn is active, the message generated should be enqueued in Conn's queue
            uint16_t message = GenerateConnMessage();
            TransmitEnqueue(CONN, message);
          }
          if(gameStatus->SonarActive) { // if Sonar is active, the message generated should be enqueued in Sonar's queue
            uint32_t message = GenerateSonarMessage();
            TransmitEnqueue(SONAR, message);
          }
          if(gameStatus->TorpedoActive) { // if Torpedo is active, the message generated should be enqueued in Torpedo's queue
            uint32_t message = GenerateTorpedoMessage();
            TransmitEnqueue(TORPEDO, message);
          }
        }
        gameStatus->hasMoveOnAlreadyon = false;// ensure that our manual override is deactivated
      } else if (gameStatus->isItMyOnAlreadyon && CurrentStatus == Play) { // if we have heard only NEXT_TURNS
        gameStatus->hasMoveOnAlreadyon = true;// move on
      } else { // we didn't transition
        printf("%");
      }
      gameStatus->NumMovesOnIsAlreadyon = 0;// reset number of move ons in a row, so we don't immediately retrigger
      gameStatus->ExpectedNextSlotNumber = 0;// reset the expected next slot number
      gameStatus->isItMyOnAlreadyon = false;// clear the next turn criteria
    }
  }

****************************************************************************
Function
  setCurrentStatus

Parameters
  CurrentGameStatus_t status

Returns
  Nothing

Description
  Sets the CurrentStatus variable to the status of the game
Notes

Author
  Holly Francis, 05/25/2020
****************************************************************************
void setCurrentStatus(CurrentGameStatus_t status) {
  CurrentStatus = status;
}

****************************************************************************
Function
  getCurrentStatus

Parameters
  None

Returns
  Nothing

Description
  Returns the CurrentStatus for use in DataParseService and TXService
Notes

Author
  Holly Francis, 05/25/2020
****************************************************************************
CurrentGameStatus_t getCurrentStatus(void){
  return CurrentStatus;
```