Network Clue Game

Gaurav Gupta, Khalef Hosany, Sampada Sonalkar, Thomas Mwakibinga

CSEE 4840 Embedded System Design

Spring 2007

Table of Contents

1. Introduction

CLUE is a crime fiction board game that tests the analytical skills of the players. The game is set in a mansion represented by a board divided into different rooms. The players each represent a character staying at this house as a guest. The owner of the mansion has been murdered and the aim of the game is to solve the crime scenario

Players take on the role of the suspects and attempt obtain the combination of the suspect, the weapon and the murder room and this effectively re-enacts the murder scene. There are 6 weapons, 6 suspects and 9 rooms.

At the beginning of play, three cards—one Suspect, one Weapon, and one Room card—are chosen at random and put into a special envelope, so that no one can see them. These cards represent the true facts of the case. The remaining cards are distributed among the players and the game can be played by 3-6 people at a time. The choice of clue was an interesting one for this project as it involves the need for a UI and a way for users to see their cards without other players seeing them. Ideally thus, this game needs to be played on different workstations and brings about the concepts of having both networking and video display capabilities on an embedded system

The aim of the game is to deduce the details of the murder—that is, the cards in the envelope. This is done by announcing suggestions to other players. An example of a suggestion is, "*I suggest it was Mrs. White, in the Library, with the Rope.*" All elements contained in the suggestion are moved into the suggested room (so in this case Mrs. White and the Rope would be moved to the Library).

The other players must then disprove the suggestion, if they can. This is done in clockwise order around the board. A suggestion is disproved by showing a card containing one of the suggestion components to the player making the suggestion (for example, the Rope), as this proves that the card cannot be in the envelope. Showing the card to the suggesting player is done in secret so the other players may not see which card is being used to disprove the suggestion. Once a suggestion has been disproved, the player's turn ends and moves onto the next player.This thus makes the game solvable by trial and error. But the true aim of Clue is to be able to solve it as quickly as possible, and so it also involves a lot of careful thinking and inference.

2. Game Design

The design consists of a hardware module for displaying the game board and user interface on the monitor, and a software module for controlling the play and exchanging information among players.

a) Board Design

The board occupies 480x480 pixels on the screen. It is laid out as a 20x20 grid of tiles. Each tile has size 24x24 pixels. Figure 1 shows the layout of rooms and pathways on the board. It also includes information on location of doors and secret passageways.

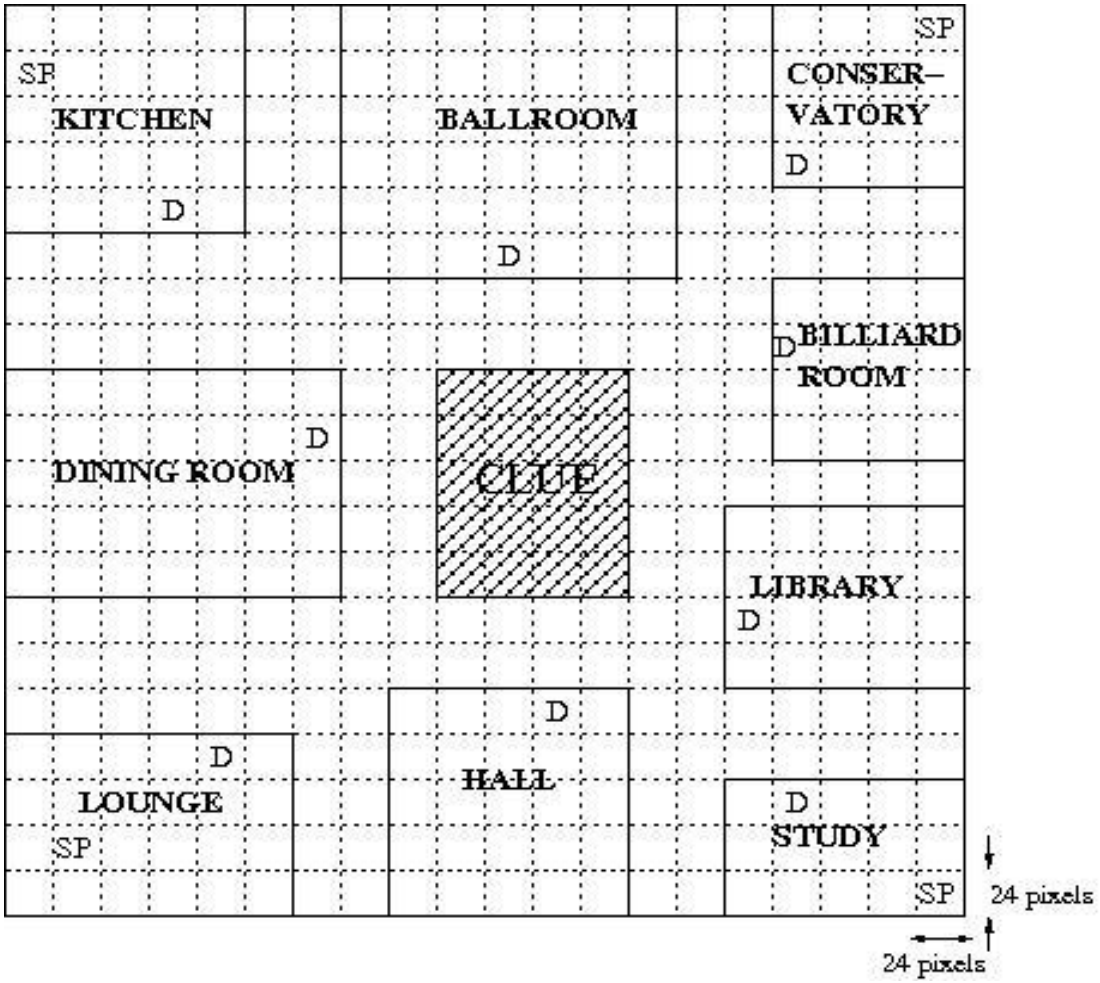

Figure 1 Board layout

b) User Interface Design

The user interface area is a 160x480 pixel area on the right side of the game board in the display. The interaction is with the help of the Enter and I (up), J (left), K (right), and M (down) keys on the keyboard. The layout of the UI is as shown in Figure 2.

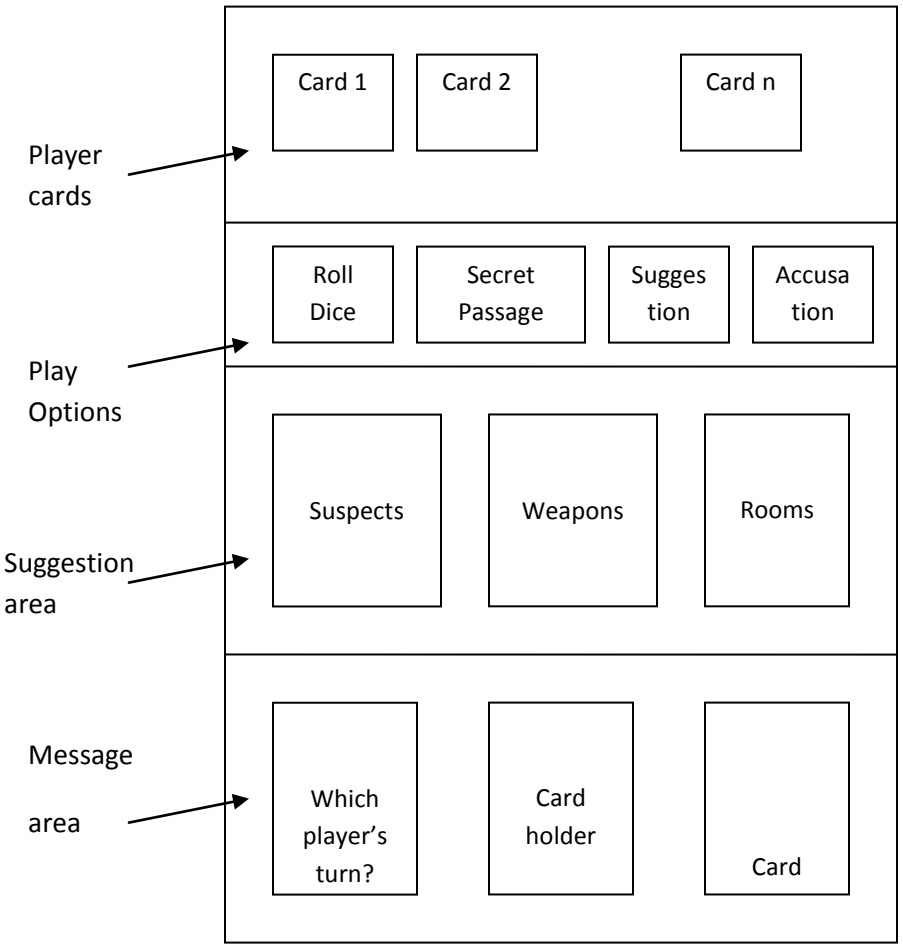

Figure 2 User interface layout

The top most area displays the player's cards and an addition "No card" space.

The play options area displays the four play options.

The suggestion area has a list of all suspects, weapons, and rooms to pick from during suggestion, disproval and accusation.

The message area displays suggestions/disprovals of other players.

3. Software design

The organization of the game logic is as shown in Figure 3. The game logic controls the game play and interfaces the keyboard, video display and the networking components.

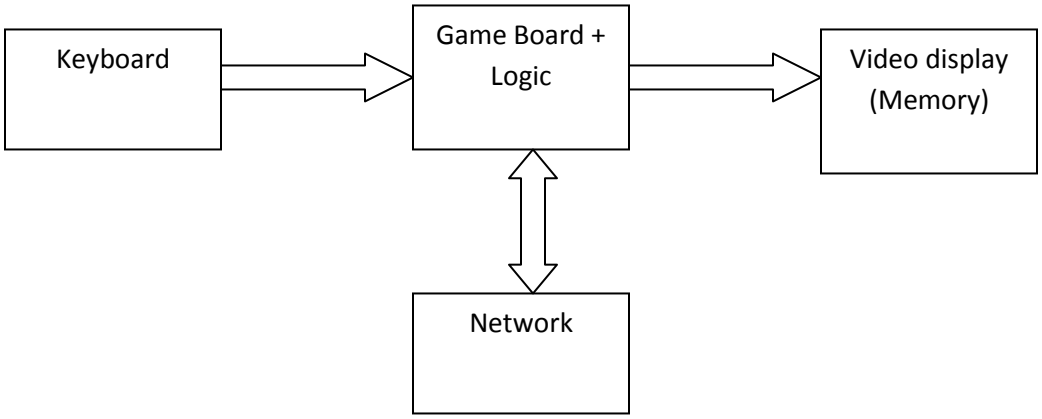

Figure 3 Software Organization

The game logic needs to perform the following tasks

1. Initial setup:
    a. The game board is a 20*20 tile board with 9 rooms connected by passages.
    b. The game logic displays the sign-in screen and allows the player to select a character for play. It also sets up the turns for players.
    c. It randomly picks a suspect card, weapon card, and room card and puts it away in the confidential case file.
    d. The remaining cards are evenly distributed among players
    e. The weapons are placed in different rooms and characters are positioned in the start position.
2. Game play: The game logic enforces the following rules of play:
    a. The players take turns to roll the dice. The game logic determines who plays next.
    b. When the dice is rolled a random number between 2-12 is generated by the system.
    c. The player may move the number of squares using horizontal, vertical, forward or backward moves; diagonal moves are not allowed. The player is not allowed to enter the same square twice on the same turn. The player is not allowed to enter on a square occupied by another character pawn.
    d. The player is allowed to enter a room only through an open door.

e. The player cannot enter a room if its doorway is blocked by another character pawn. Also, the player cannot leave a room if the doorway is blocked by another pawn.
f. The player cannot re-enter a room on the same turn.
g. On entering a room, the player can make a suggestion by moving a Suspect and Weapon into that room.
h. After the suggestion is made, the opponents try to disprove the suggestion by showing a card from their hand.
i. In the next round the player may choose to roll the dice and exit the room or to take the secret passageway to another room.
j. If a player's character pawn was moved to a room while making a suggestion, the player could choose to make a suggestion from that room or roll the dice and exit the room.

3. Making an Accusation: A player can make an accusation on his turn for a Suspect, Weapon and Room without going to that room. The system will show the confidential cards to the player. If the accusation is true, the game ends and the player wins. If the accusation is false, the player cannot play any more but can participate in disproving suggestions.

We have designed the following data structures for implementing the rules of game play. The data structures might have some additions as the details get added.

- Game Board: A 20X20 matrix of tiles
- Struct Tile: Information about a tile in the board
  - Type: Room, Pathway, Invalid
  - isDoor: Is this a door tile?
- Struct Suspect[6]: Information about a pawn on the board
  - Name
  - Position: (x,y) position on the board
  - Room information: current location and previous visited room
  - Sprite address: Information required for controlling movement of the pawn through hardware
- Struct Weapon[8]: Information about a weapon on the board
  - Name
  - Position; (x, y) position on the board
  - Room: placed in which room
  - Sprite address: Information required for controlling movement of the pawn through hardware
- Struct Room[9]: Information about the room
  - Name
  - Door position
  - Secret passage
  - List of suspects in the room: Index into pawn array
  - List of weapons in the room: Index into weapon array

- Cards: Array of all available cards

- ConfidentialFile: 3 cards kept separately
- Struct Player[6]:
    - Id : index into the Suspects array
    - Pawn
    - List of cards
    - Playing: flag indicating whether the player is active or passive. A player is passive (s)he has made an invalid accusation and it not allowed to play anymore. The player can only disprove suggestions.
- Struct Card:
    - Type: Suspect/Weapon/Room
    - Name
- Struct Position:
    - X, Y position in the Board matrix

Keyboard

The keyboard component is invoked when it is the player's turn to take some action. There are two main modes viz, play and disprove suggestion.

Play():

1. The player is given the following play options:

    a. Roll dice,

    b. Take secret passage,

    c. Make a suggestion,

    d. make an accusation.

    The player can use the J (left) and K (right) keys to navigate between options and the Enter-key to select an option.

2. When the player selects an option, the option is validated. For example, a player cannot make a suggestion if (s)he is not inside a room. The selection is sent to other players using the network.

3. If the player has selected:

    a. Roll dice: The system generates a dice number using a random number generator. The value is displayed on the 7-segment LED and transmitted to other players using the network.

    The player is allowed to move a number of steps equal to the dice value on the board using the J, K, I, M keys.

If the player enters a room (s)he can make a suggestion.

If not, the player sends a change turn packet.

b. Suggestion: Player uses J, K, I, M keys to navigate and select a suspect and weapon from the suggestion area in UI.

When the player presses 'Enter' key the suspect and weapon are moved into the room on the board. The information is also sent over the network and reflected on other boards.

c. Secret Passage: If a player is in a room that has a secret passage, the player can select this option, move to the other room and make a suggestion. This information is reflected on the video display and sent over the network for the other players.

d. Accusation: A player is allowed to make an accusation by selecting a suspect, room and weapon.

The system validates the accusation and shows the confidential file contents to the player.

If the accusation is correct, game is over.

If the accusation is incorrect, the player is switched to passive mode.

Disprove()

This method is invoked for a player who is next in turn from a player who has made a suggestion. It is also invoked when none of the preceding players are able to disprove a suggestion.

1. The player navigates his/her cards using the J, K keys.
2. When the player presses "Enter" key, the selected card is validated to check if it is one of the cards of the suggestion.
3. If the player selected "No card" there is a check for whether the player really doesn't have any of the cards in the suggsestion.
4. After the card is validated, it is sent on the network.

Game Logic software implementation maintains the current state of the game in a set of data structures described in previous sections. The Game logic has the role of receiving the network packets from the Ethernet interrupt handler and changing the state of the game board, players and weapons depending on the contents of the packet.

This is performed through logic calls to the appropriate API in video controller interface to update the registers of the video controller (hardware) with the appropriate sprite positions.

Game Logic controls the basic flow of the game by interfacing with Networking interface, keyboard interface and Video controller interface.

Following are the major functions in the Game Logic:

1. **initData( ):** This function initializes the board with all 20 x 20 tiles, suspects data structures, room data structures, cards data structures and weapons data structures. It also initializes the initial position of each of the player and weapon.

2. **initBoard( ) :** In this functions master each player first scans the master packet for first 20 seconds. If no master packet is found then player himself becomes the master and starts broadcasting 'master' packets for next 40 seconds. In these 40 seconds it also listens for any join request packets. After 40 seconds this function calls initGame() to distribute the cards, send 'GameInit' packet and start the game-play.

3. **initGame ( ) :** This function is used by master to distribute the cards among the players. It also displays its own cards on the user interface display. This function terminates by sending the 'GameInit' packet which signals the beginning of the game.

4. **UpdateDataOnPacketArrival ( ):** This function provides an interface with the network logic. Whenever a packet arrives and it is found to be a 'CLUE' packet, Ethernet interrupt handler parses the content of the packet and passes it onto the UpdateDataOnPacketArrival. This function updates the game logic data structures depending upon the contents of the packet. It has a switch statement with packet-type as control variable.

5. **sendJoinRequest ( ) :** This function is used to send a join request to master. This means master has been discovered.

6. **receivePlayerAssigned () :** This function receives player assigned by the master.

7. **sendPlayerAssigned () :** If master receives join player request it generates a player assigned packet by using this function.

8. **startGame () :** This function is used to start the game when 'GameInit' packet is received from the master.

9. **changeDice () :** This function updates dice value which thrown by other player.

10. **showMove () :** This function displays movement of player on game board.

11. **showFinalPosition () :** This function displays a player on its final position in the room.
12. **showSuggestion () :** If another player makes a suggestion this function shows that suggestion on the game board.
13. **showDisproval () :** If another player disproves a suggestion. This function is used to show that disproval on the game board.
14. **sendChangeTurnPacket() :** If a round of gameplay is over then change turn packet is generated using this function.
15. **showAccusation () :** shows accusation of other player.
16. **showPlayerOut () :** shows player out.
17. **showGameOver() :** Display game over.

4. Hardware Design

a) Networking Design

The following lists the different types of packets used by our communication protocol and will be described further below:

- Master Packet
- Join Request
- Player Assigned
- Game Init
- Dice Value
- Player Move
- Final Position
- Suggestion Packet
- Disprove Packet
- ChangeTurn packet
- Accusation packet
- Game over packet
- Player kicked out packet

Network design is based on UDP over IP. Each player sends information to other players by generating broadcast packets to all other players. This information is filtered down by the game logic based on whether or not the particular player owns that piece of information.

There is an initialization period of 60 seconds in which all players (up to a maximum of 6) are allowed to join the session. Anyone starting the game scans the network for a ‘master packet’ for the first 20 seconds. If the ‘master packet’ is not found then that terminal becomes the master and starts broadcasting ‘master packets’. Any new player joining the game senses these ‘master packets’ and sends back a new ‘player join request packet’. Master receives new player join request and reverts back with player Id in a ‘player assigned packet’. Each player Id corresponds to a unique player on the game board. This networking code does not handle player join request contention. A more sophisticated communication protocol based on unicast addressing will handle such contention.

After 60 seconds of first (master) player starting the game, the master stops answering any more join requests. He distributes the cards among the players who have joined, creates a confidential file of cards and sends this information to all other players in a ‘GameInit packet’. This marks the beginning of game play.

Master is the first player to roll the dice and make a move. Upon rolling dice a ‘Dice Roll’ packet is generated and broadcasted. The number rolled by the player is displayed in the Seven-Segment Display.

Upon making a move from one tile to another on the board a ‘Move Packet’ is generated. When a player enters a room and settles down on a tile in the room then ‘final position’ packet is generated.

After entering a room player makes a suggestion which generates the ‘Suggestion Packet’. All other players in the session receive suggestion packet and they try to disprove the suggestion in cyclic order. When a player disproves a suggestion ‘Disprove packet’ is generated. After a single round of disproval are over a player ends his turn and passes on the ‘Play’ token to next player in cycle by generating ‘Change packet’.

When a player makes an accusation then ‘accusation packet’ is generated. If accusation is proved wrong then ‘player kicked out’ packet is generated. If player’s accusation is correct then ‘game over’ packet is generated.

b) Video Controller Design

(i) Architecture

The video controller unit was implemented as shown in the diagram below:

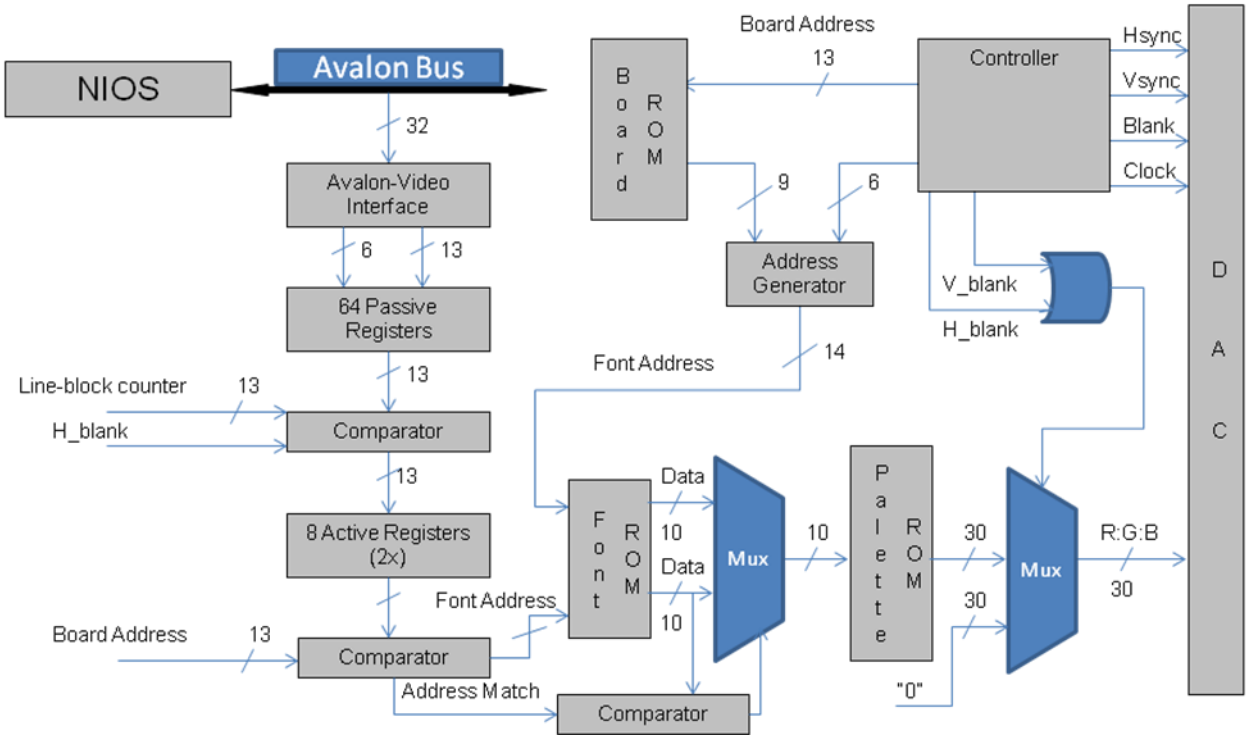

Figure 4 Video Controller Design

Because of the nature of this game, it was effective to implement a video system that was able to handle both static images (in this case the board and UI) and dynamic images(sprites). The static operation will be detailed further below.

The dynamic part of the display is implemented using sprites. At initialization, the game pieces are placed on the board by the software. This is done through communication on the Avalon Bus from the NIOS processor. The Avalon-Video interface holds a register that the software writes the sprite number and the new position to. Internally, the controller then determines which sprite needs to be moved and updates the individual register associated with that sprite.

In order to display the game properly, we determined that we would be requiring around 64 such registers. Dynamically, there are 4 places where a person can appear: on the board, in the cards area, in the message area as a suggestion/accusation and in the message area as the player

accusing/suggesting. As there are 6 players, this adds up to a total of 24 registers. Similarly weapon tiles can appear in the game area, in the cards area and in the message area and the board tiles can appear in the message area and the cards area. We thus need another 18 registers for our 6 weapons and 18 more for our 9 rooms. A couple of extra registers were also needed for the suggest tile, the accuse tile and the selector. In total, we thus needed 63 registers.

This project was an opportunity for us to learn how to design hardware and we decided to deviate from the NES video implementation proposed by professor Edwards. The main part of the change was that rather than loading shift registers during blanking period, we decided to load them in real time and perform the comparisons on the fly. The transparency implementation was something we were interested in and we kept that for the final solution. However, while most sprite-based video systems are used for a smooth display, we also decided that this was not necessary in a tile based game and we decided not to implement this and rather use a tile based approach to our controller. This made it a lot easier for us to handle the board eventually.

As a first iteration, we built the hardware for a single sprite register and then scaled it to 64. While the system performed well with one register, the additional logic of 64 created a huge propagation delay in our system. The system was clock was running at 50MHZ and gave us a maximum critical path of 20ns approximately(setup and hold times are being ignored in this case),we were seeing a delay of 74ns, allowing us to clock at only about 15MHz. This was clearly due to the atypical architecture and would result in large synchronous violations, resulting in unpredictable behavior.

We changed the video controller to make it perform better by realizing that although the screen could potentially hold 63 sprites, any line in the display would never hold more than 9. However we still had to compare 63 registers to extract the 9 ones that could be displayed. As a result, we decided to use extra registers to hold current “active” sprites and update this during blanking for the next line. To a large extent, these 9 registers then followed the same principle used by the NES controller but we were able to scale the maximum number of sprites on the screen to 64.

In the diagram above, the 63 passive registers hold the positions of every sprite on the board. The comparator is activated during horizontal blanking and compares the next line number with the position stored in the passive sprite register to search for an address match. If one is found then one of the active registers are loaded with the sprite information (sprite number; itself an index into the font ram, and position).

During the active phase, the locations of the sprites stored in the 8 active registers are compared to the current position and if there is a match the sprite information is passed as an address to the font ROM.

(ii) Video Memory

The video memory system provides a buffer interface for the controller to display the sprites appropriately. The video layout comprises of board rom, dual-ported font rom, and the pallete rom. The actual clue board is set up to fit on a 640x480 graphic display; implying that we would need a memory architecture that's of the same size with 30-bit data (10 bits for each color; red, green, blue). This technique is inefficient for 2 reasons: the memory is two big, and it doesn't take to account redundancy on pixel with the same coloring. Alternatively, using three levels of memory results in a design that is more compact and uses far less memory bits.

Board ROM

The board ROM has 4800 addresses, with each address holding 9-bits of data. Each location in this board ROM (80x60) contains only the starting address of the 8x8 pixel block in the font ROM. The controller is set to read 79 addresses. In order to display each pixel of 8x8 block the controller uses the combination of board data, pixel row count and pixel column counter to generate the input address to the font rom. This memory arrangement results in 12.5 % memory bits reduction compare to single pixel representation. Consequently, the amount of comparison logics and register are reduced significantly.

Font ROM

Because data of each address of the board rom corresponds to the initial address of particular 8x8 pixel block, it is necessary to use another rom inside the FPGA to hold and select 8x8 pixel block to be display on the screen. The controller uses the board rom data out in conjunction with the 3-bit pixel row counter and 3-bit pixel column counter as read-address to one the port to display the static sprites (background, user-interface, etc). Further, the font rom provides another read-only port for dynamics sprites, whose location on the board is constantly being updated by the software. It is important to note that sprites with repeated pattern only map to only one 8x8 memory block, eliminating data redudancy. The font rom has 16384 10-bit data, with each data information mapping to a particular color palette. Essentially, the 10-bit data coming out of the font rom becomes the read address to the palette. The font rom has two set of data coming out, which are fed into a multiplexer. The selection by the multiplexer is done by a comparator based on whether there is an address match between the static sprite and the dynamic sprite. If so, then the comparator checks for transparency on the dynamic sprite. Based on the result of this condition, the correct data from the font rom is sent to the palette.

Palette ROM

The palette rom contains all color variations that are used to make up a sprite. It has 720 colors occupying a data line of 30 bits (10 bits for red, 10 bits for blue, and 10 bits for green). The color palette ROM is arranged such that each sprite has its own block of its own color palettes. The drawback to this scheme is that it uses a little bit more memory; however, its advantages

overweigh its disadvantages. It was easier and faster to generate a color palette for a sprite (an image from the web or customable sprite) with a simple C script. The data from the palette ROM is fed into a second multiplexor. This data goes through to the DAC only when both vertical blanking and horizontal blanking are turned off and appears as a pixel on the screen

Controller

The controller serves as the brain of the video system. It sets all the appropriate flags and counters for comparators and multiplexers. It also does some arithmetic logic as well as register assignments. Furthermore, it generates the timing for HSYNC, VSYNC, and BLANKING. Ultimately, it controls the flow of the data while meeting required timing constraints.

A sample video screen shot is shown below:

Figure 5 Sample Video Screen

Logistics

a) Implementation

The video controller was developed in VHDL using the Altera's Quartus IDE. The hardware controllers for the monitor display, network controller and keyboard controller were integrated with the Nios-2 processor using the SOPC builder utility of Quartus.

The software was developed in C and compiled for running on Nios-2 processor using the Nios-2 IDE.

Concurrent version system (CVS) was used during development of the software.

The images were created using the GIMP (Gnu Image Manipulation Program). The images were built in an indexed mode using palettes of 16 colors. Using GIMP the images were saved as C header files. This created a color map array of 16 RGB values and an image array that contained indices into the color map array.

A simple C program transformed this data into memory initialization files.

b)Individual contributions

The video controller and the hardware in SOPC were built by Khalef Hosany and Thomas Mwakibinga.

The software was built by Gaurav Gupta and Sampada Sonalkar.

The images were created and transformed into memory initialization files by Sampada Sonalkar and Khalef Hosany.

The team worked together on integrating, setup and testing the system.

5. Reflections

Gaurav Gupta

Hardware-Software co-design: Before taking embedded systems course this term was a theoretical term never practiced in lab. Now I have myself designed 2 hardware-software co-designed systems. One was in lab 3 in which I moved a dice from software on a board which was implemented in hardware. Second is the big clue project where we designed network protocols, game-logic and keyboard interface in software. We designed the video controller in hardware. Then we successfully interfaced hardware and software.

Video Controller Design: While working on Clue project we learnt how to design video controller. How to design a video controller by reducing amount of data to be stored. This is done by using various levels of indexing. Color palette usage to reduce the pixel data.

Khalef Hosany

This was an extremely interesting and challenging project. The members of the "CLUE" team were eager to learn a lot from this project and beginning from the nature of the project we chose, there were several instances when we did not hesitate to try things that were somewhat uncommon in this class. Coming up with a different video controller and getting the networking at the same time as a video interface within the same project was something that several previous students of the class heavily discouraged us against. In the interest of learning, we however decided to take up the challenge. While some decisions we made were non-ideal, and later pointed out by professor Edwards, our path through architectures that we tried to redesign led us to have a better grasp on an overall embedded system than we would have had if we had followed traditional implementations. To a large extent then, we met our goals of learning a lot from this course and still came out with a working project. As far as I am concerned, this was a personally rewarding course.

As an advice to future groups (and there will be many!) learn about timing analysis early. This is something that is often ignored, as was the case for our groups. When things are scaled considerably, a lot of synchronous logic systems might break down due to timing errors and it is very hard to fix. Also, START EARLY! This has been repeated on and on but I think that it can never be emphasized enough. Hardware breaks up any time it has a chance to, and a lot of time should be spared for debugging. Also organize your work efficiently. In our case, the reason why we were able to complete the project was because we had a well-balanced group and our tasks were well distributed right from the start.

One of the important things that I take out of this project as well is how critical it is to have a good system design as early as possible. Errors that show up in the later stages of implementation become harder and harder to solve because solving something may cause another error somewhere else and this obviously scales up as the complexity of the system increases.

Also, I realized that implementing something in hardware is much harder than doing it in software!

6. Appendix

a) ***C-Code***

- **Network Data Structures**

```
typedef enum _PacketType{
  MasterBdcast,
  JoinRequest,
  PlayerAssigned,
  GameInit,
  DiceRolled,
  Move,
  FinalPosition,
  Suggestion,
  Disprove,
  ChangeTurn,
  Accusation,
  GameOver,
  PlayerOut
}PacketType;


typedef struct _Packet {

  PacketType packetType;
  unsigned int To;          // 6 means packet is addressed to
everyone
  unsigned int From;

  union {

    struct _Master {
      unsigned int id;
    }s1;

    struct _PlayerAssigned {
      unsigned int assignedId;
      unsigned char ip_address[4];
    }s2;

    struct _BoardInit {
      unsigned int playerId[21];  // playerID 6 means
confidential file
```

```
  unsigned int cardIndex[21];
  unsigned int playerCount;
}s3;

struct _Dice {
  unsigned int value;
  unsigned int playerId;
}s4;

struct _Move {
  unsigned int playerId;
  unsigned int oldPositionX;
  unsigned int oldPositionY;
  unsigned int newPositionX;
  unsigned int newPositionY;
}s5;

struct _Suggestion {
  unsigned int playerId;
  unsigned int suspectId;
  unsigned int weaponId;
  unsigned int roomId;
  unsigned int suspectX;
  unsigned int suspectY;
  unsigned int weaponX;
  unsigned int weaponY;
}s6;

struct _Disprove {
  unsigned int senderPlayerId;
  unsigned int receiverPlayerId;
  unsigned int cardIndex;      // 22 means no disapproval
}s7;

struct _ChangeTurn {
  unsigned int senderPlayerId;
  unsigned int nextPlayerId;
}s8;

struct _Accusation {
  unsigned int playerId;
  unsigned int suspectId;
  unsigned int weaponId;
  unsigned int roomId;
  unsigned int suspectX;
```

```
      unsigned int suspectY;
      unsigned int weaponX;
      unsigned int weaponY;
    }s9;

    struct _GameOver {
      unsigned int winnerId;
    }s10;

    struct _PlayerOut {
      unsigned int playerId;
    }s11;

  } u;

}Packet;
```

- **Networking API**

```
void networkInitialize();
unsigned int checksum( int start, int end, int PACKET_OFFSET);
void ethernet_interrupt_handler();
void sendPacket(Packet *pkt);
void sendChangeTurnPacket();
```

- **Game Logic Data Structures**

```
typedef enum _TileType {
    tilePathway,
    tileRoom,
    tileInvalid
}TileType;


typedef enum _CardType {
  cardPerson,
  cardWeapon,
  cardRoom
}CardType;

typedef struct _Position{
    short int x;
    short int y;
}Position;
```

```
typedef struct _Room{
    char name[NAME_SIZE_MAX];
        Position topLeft;
        Position bottomRight;
    Position door;
    typeIndex secretPassage;    // index to another room; 10
means no secret passage
    typeIndex suspects[6];
    typeIndex weapon;
    int spriteAddr;
}Room;

typedef struct _Tile {
    TileType tileType;
    int isDoor;
}Tile;

typedef struct _Suspect{
    char name[NAME_SIZE_MAX];
    Position nextPosition;
    Position currPosition;
    typeIndex currentLocation; // 10 no room, otherwise index
into Room
    typeIndex prevVisited;     // index into room
    int playing;
    int spriteAddr;
}Suspect;

typedef struct _Weapon{
    char name[NAME_SIZE_MAX];
    typeIndex roomIndex;
    Position position;
    int spriteAddr;
}Weapon;

typedef struct _Card{
  CardType cardType;
  typeIndex Index;   // index into Person, Room, Suspect array
  int spriteAddr;
}Card;

typedef struct _Player {
  typeIndex cards[MAX_HAND];    // shouldn't this be just
indices into Cards array??
  unsigned int cardCount;
```

```
  unsigned int Id;  // index into Suspects array
}Player;


extern Tile GameBoard[ROW_MAX][COL_MAX];
extern Card Cards[CARD_MAX];
extern Suspect Suspects[SUSPECT_MAX];
extern Player MyPlayer;
extern typeIndex ConfidentialFile[CONFIDENTIAL_FILE_SIZE];
extern Room Rooms[ROOM_COUNT];
extern Weapon Weapons[WEAPON_MAX];
extern typeIndex PlayerSuggestion[SUGGESTION_SIZE];

extern unsigned int Turn;
extern unsigned int PlayerCount;
extern unsigned int DisproveTurn;
```

- **Game Logic API**

```
void initBoard();

unsigned int getNextPlayer(unsigned int);

void sendJoinRequest();

void sendPlayerAssigned();

void receivedPlayerAssign(unsigned int assignedId);

void startGame( unsigned int playerId[CARD_MAX], unsigned int
cardIndex[CARD_MAX], unsigned int playerCount );

void changeDice( unsigned int value, unsigned int playerId);

void showMove( unsigned int playerId, Position *from, Position
*to);

void showDisproval( unsigned int sender, unsigned int
receiver, unsigned int cardIndex);

void showChangeTurn( unsigned int sender, unsigned int
nextPlayer);

void showAccusation( unsigned int suspect, unsigned int
weapon, unsigned int room);
```

```
void showGameOver( unsigned int winnerPlayerId);

void showPlayerOut( unsigned int playerId );

inline void invalidateSuggestion();

void initGame();

void sendChangeTurnPacket();

void showSuggestion(unsigned int playerId, unsigned int
suspect, unsigned int weapon, unsigned int room);

void showFinalPosition(unsigned int playerId, Position *from,
Position *to);

void updateDataOnPacketArrival(Packet *pkt);
```

- **Keyboard API**

```
int initialize_keyboard();

void disprove();

void play();

int validate_card(typeIndex);

int validate_mode(typeIndex);

int player_inside_room();

void make_suggestion();

void make_accusation(typeIndex[]);

void show_confidential_cards();

void walk_pathway(int);

int validate_position(Position, Position[], int);

Position finalPositionInRoom(Room);

void move_weapon_to_room(typeIndex,typeIndex);

void move_suspect_to_room(typeIndex, typeIndex);

int validate_accusation(typeIndex[]);
```

- **Video controller access data structure and API**

```
#define CARD_BASE 21

#define ACCUSE_BASEX 21

#define ACCUSE_BASEY 18

#define PLAY_OPTION_BASEX 20

#define PLAY_OPTION_BASEY 4

#define SUGGESTION_BASEX 21

#define SUGGESTION_BASEY 6

#define CARD_OFFSET 0

#define ACCUSE_MSG_OFFSET 1

#define BD_OFFSET 2

#define MY_MSG_OFFSET 3


#define PLUM_B 0

#define WHITE_B 4

#define GREEN_B 8

#define PEACOCK_B 12

#define SCARLETT_B 16

#define MUSTARD_B 20

#define PIPE_B 24

#define WRENCH_B 27

#define ROPE_B 30

#define REVOLVER_B 33

#define KNIFE_B 36

#define CANDLESTICK_B 39

#define KITCHEN_B 42

#define BALLROOM_B 44
```

```
#define CONSERVATORY_B 46
#define DINING_B 48
#define BILLIARD_B 50
#define LOUNGE_B 52
#define HALL_B 54
#define LIBRARY_B 56
#define STUDY_B 58

#define ACCUSATION_B 60
#define SUGGESTION_B 61
void writeSpriteToSram(int spriteAddr, int x, int y);
```

Code listing: Scripts for initializing memory

- **Generating Board RAM, Font RAM and Palette RAM**

```
/* Program to Generate the Board RAM
   Author: Khalef Hosany
*/

#include <stdio.h>

int main()
    {
    int array[20][26]={ 9, 9, 9, 9, 9, 0, 0,18,18,18,18,18,18,18, 0,
0,27,27,27,27,18,18,18,18,18,18,
                         9, 9, 9, 9, 9, 0, 0,18,18,18,18,18,18,18, 0,
0,27,27,27,27,18,18,18,18,18,18,
                         9, 216, 9, 9, 9, 0, 0,18,18,18,9,18,18,18,
0, 0,27,216,9,27,18,18,18,18,18,18,
                         9, 9, 9, 9, 9, 0, 0,18,18,18,18,18,18,18, 0,
0,27,27,27,27,18,18,18,18,18,18,
                         9, 9, 9, 9, 9, 0, 0,18,18,18,18,18,18,18, 0,
0, 0, 0, 0, 0,207,216,225,234,270,279,
                         0, 0, 0, 0, 0, 0, 0,18,18,18,18,18,18,18, 0,
0, 0, 0, 0, 0,18,18,18,18,18,18,
                         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,54,54,54,54,18,99,18,153,18,9,
                         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,54,54,54,54,18,108,18,162,18,18,
                        36,36,36,36,36,36,36, 0, 0,45,45,45,45, 0, 0,
0,54,324,54,54,18,117,18,171,18,27,
                        36,36,36,36,36,36,36, 0, 0,45,45,45,45, 0, 0,
0,54,54,54,54,18,126,18,180,18,36,
                        36,36,36,36,36,36,36, 0, 0,45,45,45,45, 0, 0,
0, 0, 0, 0, 0,18,135,18,189,18,45,
                        36,36,36,36,36,36,36, 0, 0,45,45,45,45, 0,
0,81,81,81,81,81,18,144,18,198,18,54,
                        36,36,36,36,36,36,36, 0, 0,45,45,45,45, 0,
0,81,81,351,81,81,18,18,18,18,18,63,
                         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,81,81,81,81,81,18,18,18,18,18,72,
                         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,81,81,81,81,81,18,18,18,18,18,80,
                         0, 0, 0, 0, 0, 0, 0, 0,72,72,72,72,72, 0, 0,
0, 0, 0, 0, 0,18,18,18,18,18,89,
                        63,63,63,63,63,63, 0, 0,72,72,72,72,72, 0, 0,
0, 0, 0, 0, 0,18,18,18,18,18,18,
                        63,216,63,63,63,63, 0, 0,72,72,72,72,72, 0,
0,90,90,90,90,90,18,18,18,18,18,18,
                        63,63,63,63,63,63, 0, 0,72,72,72,72,72, 0,
0,90,90,90,216,90,18,18,18,18,18,18,
```

```
                        63,63,63,63,63,63, 0, 0,72,72,72,72,72, 0,
0,90,90,90,90,90,18,18,18,18,18,18};


    FILE *file1;
    int offset=0;
    int counter=0;
    int i=0;
    int j=0;
    file1=fopen("board.mif","w");


fprintf(file1,"WIDTH=9;\nDEPTH=4800;\nADDRESS_RADIX=UNS;\nDATA_RADIX
=UNS;\nCONTENT BEGIN\n");


    while(i!=20)
    {
        while(offset!=3)
        {
        while(j!=26)
                        {

fprintf(file1,"\t%d\t:\t%d;\n\t%d\t:\t%d;\n\t%d\t:\t%d;\n",counter,a
rray[i][j]+(offset*3),counter+1,array[i][j]+(offset*3)+1,counter+2,a
rray[i][j]+(offset*3)+2);
                        j++;
                        counter=counter+3;
                        }

fprintf(file1,"\t%d\t:\t%d;\n\t%d\t:\t%d;\n",counter,18,counter+1,18
);
                        counter=counter+2;
                        offset++;
                        j=0;
        }

        offset=0;
    i++;
    }
    fprintf(file1,"END;\n");
    fclose(file1);


    puts("Done! press any key to continue.");
    getchar();
```

```
    return 0;
    }

/**
 * Small program to generate MIF file from the header file of an
image
 * Author: Sampada Sonalkar March 28, 2007
 **/
#include <stdio.h>
#include "clue_E.h"

#define PALETTE_MIF "mifs/clue_E_name_palette.mif"
#define PALETTE_BASE_ADDRESS (16*44)
#define PALETTE_WIDTH 30
#define PALETTE_DEPTH 16

#define FONT_RAM_MIF "mifs/clue_E_name_font_ram.mif"
#define FONT_RAM_BASE_ADDRESS (576*44)
#define FONT_RAM_WIDTH 10
#define FONT_RAM_DEPTH 576

#define ROWS 3
#define COLUMNS 3
#define BLOCK_SIZE 8

int main()     {
    FILE *fp1, *fp2;
    int i, j, k, l, m, index,value;
    unsigned long r,g,b,rgb;

    fp1 = fopen(PALETTE_MIF,"w");
    fp2 = fopen(FONT_RAM_MIF,"w");

    fprintf(fp1,"-- This is an automatically generated memory
initialization file\n");
    fprintf(fp1,"-- for loading palette ram\n\n");

    fprintf(fp2,"-- This is an automatically generated memory
initialization file\n");
    fprintf(fp2,"-- for loading font ram\n\n");

    if((fp1==NULL)||(fp2==NULL)) {
           printf("File open error\n");
           exit(1);
    }

    // Creating the palette file
    fprintf(fp1, "WIDTH=%d;\n",PALETTE_WIDTH);
```

```
    fprintf(fp1, "DEPTH=%d;\n\n",PALETTE_DEPTH);

    fprintf(fp1, "ADDRESS_RADIX=UNS;\n");
    fprintf(fp1, "DATA_RADIX=UNS;\n\n");

    fprintf(fp1, "CONTENT BEGIN\n");

    // formatting rgb as 00r00g00b
    for(i=0;i<PALETTE_DEPTH;i++) {
           r=(unsigned long)header_data_cmap[i][0];
           g=(unsigned long)header_data_cmap[i][1];
           b=(unsigned long)header_data_cmap[i][2];
           rgb=(unsigned long)((r<<22)+(g<<12)+(b<<2));
           fprintf(fp1, "    %d    :
%lu;\n",(PALETTE_BASE_ADDRESS+i),rgb);
    }

    fprintf(fp1, "END;");

    // Creating the font ram file
    fprintf(fp2, "WIDTH=%d;\n",FONT_RAM_WIDTH);
    fprintf(fp2, "DEPTH=%d;\n\n",FONT_RAM_DEPTH);

    fprintf(fp2, "ADDRESS_RADIX=UNS;\n");
    fprintf(fp2, "DATA_RADIX=UNS;\n\n");

    fprintf(fp2, "CONTENT BEGIN\n");

    /* there are 9 8x8 pixel blocks in a tile of size 24x24
     * the pixel values are stored as inclue_E_names in the palette
ram
     * the font ram is organized as block 0, block 1, ... and so on
     * the block is organized line-by-line and pixel-by-pixel
     **/
    i=0;
    for(j=0;j<ROWS;j++)                              // row
           for(k=0;k<COLUMNS;k++)                          // column
                  for(l=0;l<BLOCK_SIZE;l++)                // line
                         for(m=0;m<BLOCK_SIZE;m++)     {     //
pixel

index=j*(BLOCK_SIZE*BLOCK_SIZE*ROWS)+k*BLOCK_SIZE+l*(BLOCK_SIZE*COLU
MNS)+m;
                                value=((unsigned
int)PALETTE_BASE_ADDRESS+header_data[index]);
                                fprintf(fp2, "    %d    :
%lu;\n",(FONT_RAM_BASE_ADDRESS+i),value);
//                              printf("%d :
%d\n",i,(j*(8*8*3)+k*8+l*(8*3)+m));
```

```
                                    i++;
                                }

        fprintf(fp2, "END;");

        fclose(fp1);
        fclose(fp2);
        return 0;
    }
```

- **Generating Board RAM**

Header files

```
/*
 * CSEE 4840 CLUE
 * Authors: Gaurav Gupta and Sampada Sonalkar
 * Members: Sampada, Khalef, Thomas, Gaurav
 *
 */


#ifndef BASICTYPES_H_
#define BASICTYPES_H_

#include "system.h"
#include <io.h>

#define ROW_MAX 20
#define COL_MAX 20
#define NAME_SIZE_MAX 15
#define MAX_HAND 21
#define MASTER 0
#define CONFIDENTIAL_FILE_SIZE 3
#define ROOM_COUNT 9
#define CARD_MAX 21
#define SUSPECT_MAX 6
#define WEAPON_MAX 6
#define SUGGESTION_SIZE 3
#define INVALID_PLAYER_ID 6
#define INVALID_CARD_ID 21
#define INVALID_ROOM_ID 10
#define MASTER 0


#define BASEADDR CONTROLLER_0_BASE

#define IOWR_CONTROLLER_DATA(base, data) \
  IOWR_32DIRECT(base, 0, data)
```

```
/*
#define PATHWAY_ADDR 0
#define KITCHEN_ADDR 1
#define BALLROOM_ADDR 2
#define CONSERVATORY_ADDR 3
#define DININGROOM_ADDR 4
#define CLUE_ADDR 5
#define BILLIARD_ADDR 6
#define LOUNGE_ADDR 7
#define HALL_ADDR 8
#define LIBRARY_ADDR 9
#define STUDY_ADDR 10

#define PLUM_ADDR 11
#define WHITE_ADDR 12
#define GREEN_ADDR 13
#define PEACOCK_ADDR 14
#define SCARLETT_ADDR 15
#define MUSTARD_ADDR 16

#define PIPE_ADDR 17
#define WRENCH_ADDR 18
#define ROPE_ADDR 19
#define REVOLVER_ADDR 20
#define KNIFE_ADDR 21
#define CANDLESTICK_ADDR 22
*/

#define CARD_BASE 21
#define ACCUSE_BASEX 20
#define ACCUSE_BASEY 18
#define PLAY_OPTION_BASEX 20
#define PLAY_OPTION_BASEY 4
#define SUGGESTION_BASEX 21
#define SUGGESTION_BASEY 6

#define CARD_OFFSET 0
#define ACCUSE_MSG_OFFSET 1
#define BD_OFFSET 2
#define MY_MSG_OFFSET 3

#define PLUM_B 0
#define WHITE_B 4
#define GREEN_B 8
#define PEACOCK_B 12
#define SCARLETT_B 16
#define MUSTARD_B 20
#define PIPE_B 24
#define WRENCH_B 27
#define ROPE_B 30
#define REVOLVER_B 33
```

```
#define KNIFE_B 36
#define CANDLESTICK_B 39
#define KITCHEN_B 42
#define BALLROOM_B 44
#define CONSERVATORY_B 46
#define DINING_B 48
#define BILLIARD_B 50
#define LOUNGE_B 52
#define HALL_B 54
#define LIBRARY_B 56
#define STUDY_B 58

#define ACCUSATION_B 60
#define SUGGESTION_B 61

#define BGND_B1 62

typedef int typeIndex;

typedef enum _TileType {
    tilePathway,
    tileRoom,
    tileInvalid
}TileType;


typedef enum _CardType {
  cardPerson,
  cardWeapon,
  cardRoom
}CardType;

typedef struct _Position{
    short int x;
    short int y;
}Position;

typedef struct _Room{
    char name[NAME_SIZE_MAX];
        Position topLeft;
        Position bottomRight;
    Position door;
    typeIndex secretPassage;    // index to another room; 10
means no secret passage
    typeIndex suspects[6];
    typeIndex weapon;
    unsigned int spriteAddr;
}Room;

typedef struct _Tile {
    TileType tileType;
    int isDoor;
```

```
}Tile;

typedef struct _Suspect{
    char name[NAME_SIZE_MAX];
    Position nextPosition;
    Position currPosition;
    typeIndex currentLocation; // 10 no room, otherwise index
into Room
    typeIndex prevVisited;     // index into room
    int playing;
    unsigned int spriteAddr;
}Suspect;

typedef struct _Weapon{
    char name[NAME_SIZE_MAX];
    typeIndex roomIndex;
    Position position;
    unsigned int spriteAddr;
}Weapon;

typedef struct _Card{
  CardType cardType;
  typeIndex Index;   // index into Person, Room, Suspect array
  int spriteAddr;
}Card;

typedef struct _Player {
  typeIndex cards[MAX_HAND];        // shouldn't this be just
indices into Cards array??
  unsigned int cardCount;
  unsigned int Id;  // index into Suspects array
}Player;


extern Tile GameBoard[ROW_MAX][COL_MAX];
extern Card Cards[CARD_MAX];
extern Suspect Suspects[SUSPECT_MAX];
extern Player MyPlayer;
extern typeIndex ConfidentialFile[CONFIDENTIAL_FILE_SIZE];
extern Room Rooms[ROOM_COUNT];
extern Weapon Weapons[WEAPON_MAX];
extern typeIndex PlayerSuggestion[SUGGESTION_SIZE];

extern unsigned int Turn;
extern unsigned int PlayerCount;
extern unsigned int DisproveTurn;

void writeSpriteToSram(unsigned int spriteAddr, int x, int y);
unsigned int getRandom(unsigned int start, unsigned int end);
void initBoard();
```

```
/*
First player will be the master.
He will wait for 60 seconds for all players to join.
After 60 seconds he will distribute all the cards.

Protocol:
1. Scan network for 5 seconds for CLUE packets.
2. If no CLUE packets then I am the master
3. If there are any packets then send a JOIN packet master,
master will send me the next available player.
4. If not master wait for rest of the initialization period.

Different Packets we need :

1. MASTER
2. JOIN REQUEST
3. PLAYER
4. CARD INIT packet: next<id, list indices into Cards array>
GAME starts now
ASK professor about packet loss
5. DICE packet  <player id, dice value>
6. MOVE packets <player id, new location>
6.5. FINAL POSITION packet.
7. SUGGESTION packet <player id, suspect, weapon, room>
8. DISPROVE packet  <sender player id, reciever player id, card
index or -1 >
FIXME: we need message data structure for displaying messages.
9. CHANGETURN packet <sender player id, next player id>
10. ACCUSATION
11. GAME OVER  < player id, -1 >
12. PLAYER KICKED out packet.
*/

#endif /*BASICTYPES_H_*/

#ifndef H_GAMELOGIC_H
#define H_GAMELOGIC_H

void dummy();

void initBoard();

unsigned int getNextPlayer(unsigned int);

void sendJoinRequest();

void sendPlayerAssigned();

void receivedPlayerAssign(unsigned int assignedId);

void startGame( unsigned int playerId[CARD_MAX], unsigned int
cardIndex[CARD_MAX], unsigned int playerCount );
```

```
void changeDice( unsigned int value, unsigned int playerId);

void showMove( unsigned int playerId, Position *from, Position
*to);

void showDisproval( unsigned int sender, unsigned int receiver,
unsigned int cardIndex);

void showChangeTurn( unsigned int sender, unsigned int
nextPlayer);

void showAccusation(unsigned int playerId, unsigned int suspect,
unsigned int weapon, unsigned int room);

void showGameOver( unsigned int winnerPlayerId);

void showPlayerOut( unsigned int playerId );

inline void invalidateSuggestion();

void initGame();

void sendChangeTurnPacket();

void showSuggestion(unsigned int playerId, unsigned int suspect,
unsigned int weapon, unsigned int room);

void showFinalPosition(unsigned int playerId, Position *from,
Position *to);

void updateDataOnPacketArrival(Packet *pkt);

extern void sendPacket(Packet *pkt);
extern void move_weapon_to_room(typeIndex,typeIndex);
extern void move_suspect_to_room(typeIndex, typeIndex);
extern void disprove();
extern void play();


#endif //H_GAMELOGIC_H


#ifndef H_KEYBOARD_H
#define H_KEYBOARD_H

#include "basictypes.h"

#define BLINK_DELAY 100000

int initialize_keyboard();
void disprove();
```

```
void play();
int validate_card(typeIndex);
int validate_mode(typeIndex);
int player_inside_room();
void make_suggestion();
void make_accusation(typeIndex[]);
void show_confidential_cards();
void walk_pathway(int);
int validate_position(Position, Position[], int);
Position finalPositionInRoom(Room);
void move_weapon_to_room(typeIndex,typeIndex);
void move_suspect_to_room(typeIndex, typeIndex);
int validate_accusation(typeIndex[]);

#endif
#ifndef H_NETINTERFACE_H
#define H_NETINTERFACE_H

typedef enum _PacketType{
  MasterBdcast,
  JoinRequest,
  PlayerAssigned,
  GameInit,
  DiceRolled,
  Move,
  FinalPosition,
  Suggestion,
  Disprove,
  ChangeTurn,
  Accusation,
  GameOver,
  PlayerOut
}PacketType;


typedef struct _Packet {

  PacketType packetType;
  unsigned int To;         // 6 means packet is addressed to
everyone
  unsigned int From;

  union {

    struct _Master {
      unsigned int id;
    }s1;

    struct _PlayerAssigned {
      unsigned int assignedId;
      unsigned char ip_address[4];
    }s2;
```

```
    struct _BoardInit {
      unsigned int playerId[21];  // playerID 6 means
confidential file
      unsigned int cardIndex[21];
      unsigned int playerCount;
    }s3;

    struct _Dice {
      unsigned int value;
      unsigned int playerId;
    }s4;

    struct _Move {
      unsigned int playerId;
      unsigned int oldPositionX;
      unsigned int oldPositionY;
      unsigned int newPositionX;
      unsigned int newPositionY;
    }s5;

    struct _Suggestion {
      unsigned int playerId;
      unsigned int suspectId;
      unsigned int weaponId;
      unsigned int roomId;
      unsigned int suspectX;
      unsigned int suspectY;
      unsigned int weaponX;
      unsigned int weaponY;
    }s6;

    struct _Disprove {
      unsigned int senderPlayerId;
      unsigned int receiverPlayerId;
      unsigned int cardIndex;      // 22 means no disapproval
    }s7;

    struct _ChangeTurn {
      unsigned int senderPlayerId;
      unsigned int nextPlayerId;
    }s8;

    struct _Accusation {
      unsigned int playerId;
      unsigned int suspectId;
      unsigned int weaponId;
      unsigned int roomId;
      unsigned int suspectX;
      unsigned int suspectY;
      unsigned int weaponX;
      unsigned int weaponY;
```

```
    }s9;

    struct _GameOver {
      unsigned int winnerId;
    }s10;

    struct _PlayerOut {
      unsigned int playerId;
    }s11;

  } u;

}Packet;

void networkInitialize();

unsigned int checksum( int start, int end, int PACKET_OFFSET);

void ethernet_interrupt_handler();

void sendPacket(Packet *pkt);

void sendChangeTurnPacket();

#endif //H_NETINTERFACE_H
```

C-Files

```
/*
 * CSEE 4840 CLUE
 * Authors: Gaurav Gupta and Sampada Sonalkar
 * Members: Sampada, Khalef, Thomas, Gaurav
 *
 */

#include "basic_io.h"
#include "DM9000A.h"
#include <alt_types.h>
#include "alt_up_ps2_port.h"
#include "ps2_keyboard.h"
//#include "ps2_mouse.h"
#include "LCD.h"

#include "basictypes.h"
#include "keyboard.h"
#include "netinterface.h"
#include "gamelogic.h"
```

```
#include <ctype.h>

int main()
{

  // Initialize the LCD and display a welcome message
  LCD_Init();
  LCD_Show_Text("4840 CLUE");

  // Clear the LEDs to zero (will display interrupt count)
  outport(SEG7_DISPLAY_BASE, 0);

  // Print a friendly welcome message
  printf("4840 CLUE started\n");

  networkInitialize();

  // Initialize the keyboard
  printf("Please wait three seconds to initialize
keyboard\n");
  clear_FIFO();

  initialize_keyboard();
  initBoard();

  for(;;)
  {

    if(MyPlayer.Id==Turn && DisproveTurn ==
INVALID_PLAYER_ID)
        play();

        if(DisproveTurn == MyPlayer.Id)
            disprove();

    // wait for ethernet and keyboard interrupts
  }

  return 0;

}
#include "basic_io.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> //for usleep()
#include <string.h>
#include "basictypes.h"
```

```
#include "netinterface.h"
#include "keyboard.h"
#include "gamelogic.h"

#define SECOND 1000000
#define WAIT_FOR_MASTER 10
#define WAIT_FOR_PLAYERS 20

static int nextAvailablePlayer = 1;

 Tile GameBoard[ROW_MAX][COL_MAX];
 Card Cards[CARD_MAX];
 Suspect Suspects[SUSPECT_MAX];
 Player MyPlayer;
 typeIndex ConfidentialFile[CONFIDENTIAL_FILE_SIZE];
 Room Rooms[ROOM_COUNT];
 Weapon Weapons[WEAPON_MAX];
 typeIndex PlayerSuggestion[SUGGESTION_SIZE];

 unsigned int Turn;
 unsigned int PlayerCount;
 unsigned int DisproveTurn;

void dummy()
{ return ;}


void writeSpriteToSram(unsigned int spriteAddr, int x, int
y)
{
    alt_u16 sprite_addr = spriteAddr;
    alt_u16 position;
    alt_u32 data;

    if(x==-1 && y==-1)  {   //remove sprite from screen
        position = 5000;
    }
    else    {
        position = 3 * (y * 80 + x);
    }
    data = (sprite_addr << 16)|(position);

    IOWR_CONTROLLER_DATA( BASEADDR, data );
    usleep(10);
    IOWR_CONTROLLER_DATA( BASEADDR, 0xffffffff );
    usleep(10);
```

```
}

unsigned int getNextPlayer(unsigned int player)
{
  int ret = player;
  do{
    if(ret<PlayerCount-1)
      ret++ ;
    else
      ret = 0;

  if (ret == Turn)
    return INVALID_PLAYER_ID;

  }while(!Suspects[ret].playing);

  return ret;
}

unsigned int getRandom(unsigned int start, unsigned int
end) {

  unsigned int ret = rand() % (end -start +1);

  if(ret < start )
    return ret + start;
  else
    return ret;
}

void initData(){

printf("middle initData %d %d " , ROW_MAX, COL_MAX);

  int i,j;
  for( i=0; i<ROW_MAX ; ++i) {
    for( j=0; j<COL_MAX; ++j) {
      if( i>=0 && i<=4 && j>=0 && j<=4){
        //kitchen
        GameBoard[i][j].tileType = tileRoom;
        continue;
      }
      if( i>=0 && i<=5 && j>=7 && j<=13){
        //Ballroom
        GameBoard[i][j].tileType = tileRoom;
        continue;
      }
```

```
    if( i>=0 && i<=3 && j>=16 && j<=19){
      //conservatory
      GameBoard[i][j].tileType = tileRoom;
      continue;
    }
    if( i>=8 && i<=12 && j>=0 && j<=6){
      //Dining Room
      GameBoard[i][j].tileType = tileRoom;
      continue;
    }
    if( i>=16 && i<=19 && j>=0 && j<=5){
      //Lounge
      GameBoard[i][j].tileType = tileRoom;
      continue;
    }
    if( i>=15 && i<=19 && j>=8 && j<=12){
      //Hall
      GameBoard[i][j].tileType = tileRoom;
      continue;
    }
    if( i>=6 && i<=9 && j>=16 && j<=19){
      //billiard room
      GameBoard[i][j].tileType = tileRoom;
      continue;
    }
    if( i>=11 && i<=14 && j>=15 && j<=19){
      //library
      GameBoard[i][j].tileType = tileRoom;
      continue;
    }
    if( i>=17 && i<=19 && j>=15 && j<=19){
      //study
      GameBoard[i][j].tileType = tileRoom;
      continue;
    }
   if( i>=8 && i<=12 && j>=9 && j<=12){
      GameBoard[i][j].tileType = tileInvalid;
      continue;
    }
  GameBoard[i][j].tileType  = tilePathway;
  }
}


strcpy( Suspects[0].name, "PLUM");
Suspects[0].currPosition.x = 6;
Suspects[0].currPosition.y = 0;
```

```
  Suspects[0].nextPosition.x = -1;
  Suspects[0].nextPosition.y = -1;
  Suspects[0].currentLocation = INVALID_ROOM_ID;
  Suspects[0].prevVisited = INVALID_ROOM_ID;
  Suspects[0].playing = 0;
  Suspects[0].spriteAddr = PLUM_B;
  writeSpriteToSram( PLUM_B + BD_OFFSET,
Suspects[0].currPosition.x, Suspects[0].currPosition.y  );

  strcpy( Suspects[1].name, "WHITE");
  Suspects[1].currPosition.x = 15;
  Suspects[1].currPosition.y = 0;
  Suspects[1].nextPosition.x = -1;
  Suspects[1].nextPosition.y = -1;
  Suspects[1].currentLocation = INVALID_ROOM_ID;
  Suspects[1].prevVisited = INVALID_ROOM_ID;
  Suspects[1].playing = 0;
  Suspects[1].spriteAddr = WHITE_B;
  writeSpriteToSram( WHITE_B + BD_OFFSET,
Suspects[1].currPosition.x, Suspects[1].currPosition.y  );

  strcpy( Suspects[2].name, "GREEN");
  Suspects[2].currPosition.x = 19;
  Suspects[2].currPosition.y = 5;
  Suspects[2].nextPosition.x = -1;
  Suspects[2].nextPosition.y = -1;
  Suspects[2].currentLocation = INVALID_ROOM_ID;
  Suspects[2].prevVisited = INVALID_PLAYER_ID;
  Suspects[2].playing = 0;
  Suspects[2].spriteAddr = GREEN_B;
  writeSpriteToSram( GREEN_B + BD_OFFSET,
Suspects[2].currPosition.x, Suspects[2].currPosition.y  );


  strcpy( Suspects[3].name, "PEACOCK");
  Suspects[3].currPosition.x = 19;
  Suspects[3].currPosition.y = 15;
  Suspects[3].nextPosition.x = -1;
  Suspects[3].nextPosition.y = -1;
  Suspects[3].currentLocation = INVALID_ROOM_ID;
  Suspects[3].prevVisited = INVALID_PLAYER_ID;
  Suspects[3].playing = 0;
  Suspects[3].spriteAddr = PEACOCK_B;
  writeSpriteToSram( PEACOCK_B + BD_OFFSET,
Suspects[3].currPosition.x, Suspects[3].currPosition.y  );

  strcpy( Suspects[5].name, "MUSTARD");
```

```
  Suspects[5].currPosition.x = 6;
  Suspects[5].currPosition.y = 19;
  Suspects[5].nextPosition.x = -1;
  Suspects[5].nextPosition.y = -1;
  Suspects[5].currentLocation = INVALID_ROOM_ID;
  Suspects[5].prevVisited = INVALID_PLAYER_ID;
  Suspects[5].playing = 0;
  Suspects[5].spriteAddr = MUSTARD_B;
  writeSpriteToSram( MUSTARD_B + BD_OFFSET,
Suspects[5].currPosition.x, Suspects[5].currPosition.y  );

  strcpy( Suspects[4].name, "SCARLETT");
  Suspects[4].currPosition.x = 0;
  Suspects[4].currPosition.y = 15;
  Suspects[4].nextPosition.x = -1;
  Suspects[4].nextPosition.y = -1;
  Suspects[4].currentLocation = INVALID_ROOM_ID;
  Suspects[4].prevVisited = INVALID_PLAYER_ID;
  Suspects[4].playing = 0;
  Suspects[4].spriteAddr = SCARLETT_B;
  writeSpriteToSram( SCARLETT_B + BD_OFFSET,
Suspects[4].currPosition.x, Suspects[4].currPosition.y  );


  for(i=0;i<ROOM_COUNT;++i)
    for(j=0;j<SUSPECT_MAX;++j)
      Rooms[i].suspects[j]=-1;

  strcpy( Rooms[0].name, "KITCHEN");
  Rooms[0].secretPassage = 8;
//  Rooms[0].suspects;
  Rooms[0].weapon = -1;
  Rooms[0].door.x = 3;
  Rooms[0].door.y = 4;
  Rooms[0].topLeft.x = 0;
  Rooms[0].topLeft.y = 0;
  Rooms[0].bottomRight.x = 4;
  Rooms[0].bottomRight.y = 4;
  Rooms[0].spriteAddr = KITCHEN_B;

  strcpy( Rooms[1].name, "DINING ROOM");
  Rooms[1].secretPassage = -1;
  Rooms[1].weapon = -1;
  Rooms[1].door.x = 6;
  Rooms[1].door.y = 9;
  Rooms[1].topLeft.x = 0;
  Rooms[1].topLeft.y = 8;
```

```
Rooms[1].bottomRight.x = 6;
Rooms[1].bottomRight.y = 11;
Rooms[1].spriteAddr = DINING_B;

strcpy( Rooms[2].name, "LOUNGE");
Rooms[2].secretPassage = 5;
Rooms[2].weapon = -1;
Rooms[2].door.x = 4;
Rooms[2].door.y = 16;
Rooms[2].topLeft.x = 0;
Rooms[2].topLeft.y = 16;
Rooms[2].bottomRight.x = 5;
Rooms[2].bottomRight.y = 19;
Rooms[2].spriteAddr = LOUNGE_B;

strcpy( Rooms[3].name, "BALLROOM");
Rooms[3].secretPassage = -1;
Rooms[3].weapon = -1;
Rooms[3].door.x = 10;
Rooms[3].door.y = 5;
Rooms[3].topLeft.x = 7;
Rooms[3].topLeft.y = 0;
Rooms[3].bottomRight.x = 13;
Rooms[3].bottomRight.y = 5;
Rooms[3].spriteAddr = BALLROOM_B;

strcpy( Rooms[4].name, "HALL");
Rooms[4].secretPassage = -1;
Rooms[4].weapon = -1;
Rooms[4].door.x = 11;
Rooms[4].door.y = 15;
Rooms[4].topLeft.x = 8;
Rooms[4].topLeft.y = 15;
Rooms[4].bottomRight.x = 12;
Rooms[4].bottomRight.y = 19;
Rooms[4].spriteAddr = HALL_B;

strcpy( Rooms[5].name, "CONSERV");
Rooms[5].secretPassage = 2;
Rooms[5].weapon = -1;
Rooms[5].door.x = 16;
Rooms[5].door.y = 3;
Rooms[5].topLeft.x = 16;
Rooms[5].topLeft.y = 0;
Rooms[5].bottomRight.x = 19;
Rooms[5].bottomRight.y = 3;
Rooms[5].spriteAddr = CONSERVATORY_B;
```

```
  strcpy( Rooms[6].name, "BILLIARD");
  Rooms[6].secretPassage = -1;
  Rooms[6].weapon = -1;
  Rooms[6].door.x = 16;
  Rooms[6].door.y = 7;
  Rooms[6].topLeft.x = 16;
  Rooms[6].topLeft.y = 6;
  Rooms[6].bottomRight.x = 19;
  Rooms[6].bottomRight.y = 9;
  Rooms[6].spriteAddr = BILLIARD_B;

  strcpy( Rooms[7].name, "LIBRARY");
  Rooms[7].secretPassage = 1;
  Rooms[7].weapon = -1;
  Rooms[7].door.x = 15;
  Rooms[7].door.y = 13;
  Rooms[7].topLeft.x = 15;
  Rooms[7].topLeft.y = 11;
  Rooms[7].bottomRight.x = 19;
  Rooms[7].bottomRight.y = 14;
  Rooms[7].spriteAddr = LIBRARY_B;

  strcpy( Rooms[8].name, "STUDY");
  Rooms[8].secretPassage = 0;
  Rooms[8].weapon = -1;
  Rooms[8].door.x = 16;
  Rooms[8].door.y = 17;
  Rooms[8].topLeft.x = 15;
  Rooms[8].topLeft.y = 17;
  Rooms[8].bottomRight.x = 19;
  Rooms[8].bottomRight.y = 19;
  Rooms[8].spriteAddr = STUDY_B;

  strcpy( Weapons[0].name, "PIPE");
  Weapons[0].roomIndex = 0;
  Weapons[0].position.x = 1;
  Weapons[0].position.y = 2;
  Weapons[0].spriteAddr = PIPE_B;
  writeSpriteToSram( PIPE_B + BD_OFFSET,
Weapons[0].position.x, Weapons[0].position.y  );

  strcpy( Weapons[1].name, "WRENCH");
  Weapons[1].roomIndex = 1;
  Weapons[1].position.x = 0;
  Weapons[1].position.y = 9;
  Weapons[1].spriteAddr = WRENCH_B;
```

```
  writeSpriteToSram( WRENCH_B + BD_OFFSET,
Weapons[1].position.x, Weapons[1].position.y  );

  strcpy( Weapons[2].name, "ROPE");
  Weapons[2].roomIndex = 2;
  Weapons[2].position.x = 0;
  Weapons[2].position.y = 17;
  Weapons[2].spriteAddr = ROPE_B;
  writeSpriteToSram( ROPE_B + BD_OFFSET,
Weapons[2].position.x, Weapons[2].position.y  );

  strcpy( Weapons[3].name, "REVOLVER");
  Weapons[3].roomIndex = 3;
  Weapons[3].position.x = 7;
  Weapons[3].position.y = 1;
  Weapons[3].spriteAddr = REVOLVER_B;
  writeSpriteToSram( REVOLVER_B + BD_OFFSET,
Weapons[3].position.x, Weapons[3].position.y   );

  strcpy( Weapons[4].name, "KNIFE");
  Weapons[4].roomIndex = 4;
  Weapons[4].position.x = 8;
  Weapons[4].position.y = 16;
  Weapons[4].spriteAddr = KNIFE_B;
  writeSpriteToSram( KNIFE_B + BD_OFFSET,
Weapons[4].position.x, Weapons[4].position.y  );

  strcpy( Weapons[5].name, "CANDLESTICK");
  Weapons[5].roomIndex = 5;
  Weapons[5].position.x = 16;
  Weapons[5].position.y = 1;
  Weapons[5].spriteAddr = CANDLESTICK_B;
  writeSpriteToSram( CANDLESTICK_B + BD_OFFSET ,
Weapons[5].position.x, Weapons[5].position.y  );


  for(i=0; i<SUSPECT_MAX; ++i){
    Cards[i].cardType = cardPerson;
    Cards[i].Index = i;
    Cards[i].spriteAddr = Suspects[i].spriteAddr +
CARD_OFFSET;
  }
  for(i=SUSPECT_MAX; i<WEAPON_MAX+SUSPECT_MAX; ++i){
    Cards[i].cardType =cardWeapon;
    Cards[i].Index = i-SUSPECT_MAX;
    Cards[i].spriteAddr = Weapons[i-SUSPECT_MAX].spriteAddr
+ CARD_OFFSET;
```

```
  }
  for(i=SUSPECT_MAX+WEAPON_MAX; i< SUSPECT_MAX + WEAPON_MAX
+ ROOM_COUNT; ++i){
    Cards[i].cardType =cardRoom;
    Cards[i].Index = i-(SUSPECT_MAX+WEAPON_MAX);
    Cards[i].spriteAddr = Rooms[i-
(SUSPECT_MAX+WEAPON_MAX)].spriteAddr + CARD_OFFSET;
  }

  Turn = 0;
  PlayerCount = 0;
  DisproveTurn = INVALID_PLAYER_ID;
}

void initBoard(){

printf("initBoard");

  Packet *pkt=NULL;
  int count;

  initData();

  MyPlayer.Id = INVALID_PLAYER_ID;

  /*Wait for 10 sec, listen to udp packets, if no
MyPlayer.Id packet -
  then take over as master   */
  usleep( WAIT_FOR_MASTER * SECOND );

  pkt = (Packet*)malloc(sizeof(Packet));

  if( MyPlayer.Id==6) {
    MyPlayer.Id =0;
    pkt->packetType = MasterBdcast;
    pkt->To = INVALID_PLAYER_ID;
    pkt->From = MyPlayer.Id;
    pkt->u.s1.id = MyPlayer.Id;

    count = 0;
    while (count < (WAIT_FOR_PLAYERS / 1 )){
      sendPacket(pkt);
      usleep( 1 * SECOND );
      count++;
    }

    initGame();
```

```
    play();
    //  play will send the changeturn packet in case no
suggestion is made

  } /* else {
    //send JoinRequest packet
    //    pkt->packetType = JoinRequest;
    //    sendPacket(pkt);
    // now wait for assignment packet from MyPlayer.Id

    } */
    /* Enter infinite loop after this */

}

void initGame( )
{
  int cardCount=0,i,player;
  Packet *pkt;

  srand(MyPlayer.Id);

  pkt = (Packet*)malloc(sizeof(Packet));
  PlayerCount = nextAvailablePlayer;

    //FIXME one weapon,suspect,room for confidential file
    ConfidentialFile[0] = 0; //getRandom(cardCount,
CARD_MAX-1);
    pkt->u.s3.playerId[cardCount] = INVALID_PLAYER_ID;
    pkt->u.s3.cardIndex[cardCount] = ConfidentialFile[0];
    cardCount++;
    ConfidentialFile[1] = 6; //getRandom(cardCount,
CARD_MAX-1);
    pkt->u.s3.playerId[cardCount] = INVALID_PLAYER_ID;
    pkt->u.s3.cardIndex[cardCount] = ConfidentialFile[1];
    cardCount++;
    ConfidentialFile[2] = 12; //getRandom(cardCount,
CARD_MAX-1);
    pkt->u.s3.playerId[cardCount] = INVALID_PLAYER_ID;
    pkt->u.s3.cardIndex[cardCount] = ConfidentialFile[2];
    cardCount++;

    MyPlayer.cardCount =0;
    player = 0;
    for(i=1; i<6; ++i){
      if( player==0){
        MyPlayer.cards[MyPlayer.cardCount] = i;
```

```
        writeSpriteToSram(
Cards[MyPlayer.cards[MyPlayer.cardCount]].spriteAddr ,
CARD_BASE + (MyPlayer.cardCount % 5), MyPlayer.cardCount /5
+ 1 );
        MyPlayer.cardCount++;
      }
      pkt->u.s3.playerId[cardCount] = player;
      pkt->u.s3.cardIndex[cardCount] = i;
      cardCount++;

      player++;
      player = player % PlayerCount;
    }

    //    player = 0;
    for(i=7; i<12; ++i){
      if( player==0){
        MyPlayer.cards[MyPlayer.cardCount] = i;
        writeSpriteToSram(
Cards[MyPlayer.cards[MyPlayer.cardCount]].spriteAddr,
CARD_BASE + (MyPlayer.cardCount % 5), MyPlayer.cardCount /5
+1 );
        MyPlayer.cardCount++;
      }
      pkt->u.s3.playerId[cardCount] = player;
      pkt->u.s3.cardIndex[cardCount] = i;
      cardCount++;

      player++;
      player = player % PlayerCount;
    }

    //    player = 0;
    for(i=13; i<21; ++i){
      if( player==0){
        MyPlayer.cards[MyPlayer.cardCount] = i;
        writeSpriteToSram(
Cards[MyPlayer.cards[MyPlayer.cardCount]].spriteAddr,
CARD_BASE + (MyPlayer.cardCount % 5), MyPlayer.cardCount /5
+1 );
        MyPlayer.cardCount++;
      }
      pkt->u.s3.playerId[cardCount] = player;
      pkt->u.s3.cardIndex[cardCount] = i;
      cardCount++;

      player++;
```

```
      player = player % PlayerCount;
    }

  //FIXME shuffle cards after every distribution

  pkt->u.s3.playerCount = PlayerCount;
  pkt->packetType = GameInit;
  pkt->To = INVALID_PLAYER_ID;
  pkt->From = MyPlayer.Id;

  sendPacket(pkt);
  free(pkt);

  // PlayerId = 0 = MyPlayer.Id will have first turn
  Turn = 0;

  for(i=0;i<PlayerCount;++i){
    Suspects[i].playing = 1;
  }

  //update the display
}

void updateDataOnPacketArrival( Packet *pkt) {

  Position From,To;

  switch(pkt->packetType){

  case MasterBdcast:
    if( MyPlayer.Id == INVALID_PLAYER_ID ){
printf("Master found\n");
      sendJoinRequest();
    }
    break;

  case JoinRequest:
    if(MyPlayer.Id == 0){
      sendPlayerAssigned( );
    }
    break;

  case PlayerAssigned:
    //    if( PlayerAssignedIsMe(pkt->u.s2.ip_address) ) {
      /* I need IP address till Player IDs have been
assigned. */
    if(MyPlayer.Id == INVALID_PLAYER_ID){
```

```
printf("Recvd player assigned %d\n",  pkt-
>u.s2.assignedId);
       MyPlayer.Id = pkt->u.s2.assignedId;
    }
       //}
    break;

  case GameInit:

printf("game started by master\n");
//    startGame(&(pkt->u.s3.playerId), &(pkt-
>u.s3.cardIndex), &(pkt->u.s3.playerCount));
    startGame(pkt->u.s3.playerId, pkt->u.s3.cardIndex, pkt-
>u.s3.playerCount);
    break;

  case DiceRolled:

    changeDice(pkt->u.s4.value, pkt->u.s4.playerId);
    break;

  case Move:

    From.x = pkt->u.s5.oldPositionX;
    From.y = pkt->u.s5.oldPositionY;
    To.x = pkt->u.s5.newPositionX;
    To.y = pkt->u.s5.newPositionY;

    showMove(pkt->u.s5.playerId, &From, &To);
    break;

  case FinalPosition:
    //FIXME implement this!
    From.x = pkt->u.s5.oldPositionX;
    From.y = pkt->u.s5.oldPositionY;
    To.x = pkt->u.s5.newPositionX;
    To.y = pkt->u.s5.newPositionY;
    showFinalPosition(pkt->u.s5.playerId, &From, &To);
    break;

  case Suggestion:

    showSuggestion( pkt->u.s6.playerId, pkt-
>u.s6.suspectId, pkt->u.s6.weaponId, pkt->u.s6.roomId );

    DisproveTurn = getNextPlayer(pkt->From);
    // if next(packet.from) == me then call 'disprove()'
```

```
    break;

  case Disprove:

    showDisproval(pkt->u.s7.senderPlayerId, pkt-
>u.s7.receiverPlayerId, pkt->u.s7.cardIndex);
    if(Turn == MyPlayer.Id ){
      if( pkt->u.s7.cardIndex != INVALID_CARD_ID ||
          (pkt->u.s7.senderPlayerId==MyPlayer.Id -1 ||
(pkt->u.s7.senderPlayerId==PlayerCount-1 &&
MyPlayer.Id==0))){
        DisproveTurn = INVALID_PLAYER_ID;
            sendChangeTurnPacket();
      }
      else {
        DisproveTurn = getNextPlayer(pkt->From);
      }
    }
      break;

  case ChangeTurn:

    showChangeTurn(pkt->u.s8.senderPlayerId, pkt-
>u.s8.nextPlayerId);
    DisproveTurn = INVALID_PLAYER_ID;
    break;

  case Accusation:

    showAccusation(pkt->From, pkt->u.s9.suspectId, pkt-
>u.s9.weaponId, pkt->u.s9.roomId);
    break;

  case GameOver:

    showGameOver(pkt->u.s10.winnerId);
    break;

  case PlayerOut:

    showPlayerOut(pkt->u.s11.playerId);
    break;

    default : break;

  }
```

```
}

void sendJoinRequest()
{
  Packet *pkt;

  pkt = (Packet*) malloc(sizeof(Packet));
  pkt->packetType = JoinRequest;
  pkt->To = 0;
  pkt->From = INVALID_PLAYER_ID;
  sendPacket(pkt);

  free(pkt);


}

void receivedPlayerAssign(unsigned int assignedId){

    MyPlayer.Id = assignedId;
}

void sendPlayerAssigned( )
{
  Packet *pkt;

  pkt = (Packet*) malloc(sizeof(Packet));
  pkt->packetType = PlayerAssigned;
  pkt->To = INVALID_PLAYER_ID;
  pkt->From = MyPlayer.Id;
  pkt->u.s2.assignedId = nextAvailablePlayer;
  nextAvailablePlayer++;
  //  strcpy(pkt->u.s2.ip_address,  ip_address);

printf("send player assigned %d \n", pkt->u.s2.assignedId
);
  sendPacket(pkt);
  free(pkt);
}


void startGame( unsigned int playerId[CARD_MAX], unsigned
int cardIndex[CARD_MAX], unsigned int playerCount )
{
  int cardCount=0,i;
```

```
  srand(MyPlayer.Id);

  PlayerCount = playerCount;

  ConfidentialFile[0] = cardIndex[0];
  ConfidentialFile[1] = cardIndex[1];
  ConfidentialFile[2] = cardIndex[2];

  MyPlayer.cardCount =0;
  for( i=3 ; i<CARD_MAX ; ++i ){
    if(playerId[i] == MyPlayer.Id){
      MyPlayer.cards[cardCount] = cardIndex[i];
      writeSpriteToSram(
Cards[MyPlayer.cards[cardCount]].spriteAddr , CARD_BASE +
(cardCount % 5), cardCount /5 +1 );
      cardCount++;
    }
  }
  MyPlayer.cardCount = cardCount;
  // NO_CARD writeSpriteToSram(
Cards[MyPlayer.cards[cardCount]].spriteAddr, CARD_BASE +
(cardCount % 5), cardCount /5  );
  Turn =0;

  for(i=0; i<PlayerCount;++i)
    Suspects[i].playing =1;

  //update the display
}

void changeDice( unsigned int value, unsigned int playerId
)
{

  printf( "Player %s diced %d", Suspects[playerId].name,
value);

  //change display
  outport(SEG7_DISPLAY_BASE, value);


}

int getRoomId(unsigned int x, unsigned int y)
{
  int i;
```

```
  for(i=0;i<ROOM_COUNT;++i) {
    if( x>=Rooms[i].topLeft.x && x<=Rooms[i].bottomRight.x
&& y>=Rooms[i].topLeft.y && y<=Rooms[i].bottomRight.y  )
      return i;
  }
  return -1;
}

void showMove(unsigned int playerId, Position *from,
Position *to)
{

  Suspects[playerId].currPosition.x = from->x;
  Suspects[playerId].currPosition.y = from->y;
  Suspects[playerId].nextPosition.x = to->x;
  Suspects[playerId].nextPosition.y = to->y;

  writeSpriteToSram( Suspects[playerId].spriteAddr +
BD_OFFSET, Suspects[playerId].nextPosition.x,
Suspects[playerId].nextPosition.y );
  // display animation

  Suspects[playerId].currPosition.x =
Suspects[playerId].nextPosition.x;
  Suspects[playerId].currPosition.y =
Suspects[playerId].nextPosition.y;
  Suspects[playerId].nextPosition.x = -1;
  Suspects[playerId].nextPosition.y = -1;
//  Suspects[playerId].prevVisited =
Suspects[playerId].currentLocation;
//  Suspects[playerId].currentLocation =
getRoomId(Suspects[playerId].currPosition.x,Suspects[player
Id].currPosition.y);

  printf("%s moved from (%d,%d) to (%d,%d).\n",
Suspects[playerId].name, from->x, from->y, to->x, to->y);

}

void showFinalPosition(unsigned int playerId, Position
*from, Position *to)
{


  Suspects[playerId].currPosition.x = from->x;
  Suspects[playerId].currPosition.y = from->y;
  Suspects[playerId].nextPosition.x = to->x;
```

```
  Suspects[playerId].nextPosition.y = to->y;

  // display player in new position
  writeSpriteToSram( Suspects[playerId].spriteAddr +
BD_OFFSET, Suspects[playerId].nextPosition.x,
Suspects[playerId].nextPosition.y );

  Suspects[playerId].currPosition.x =
Suspects[playerId].nextPosition.x;
  Suspects[playerId].currPosition.y =
Suspects[playerId].nextPosition.y;
  Suspects[playerId].nextPosition.x = -1;
  Suspects[playerId].nextPosition.y = -1;
  Suspects[playerId].prevVisited =
Suspects[playerId].currentLocation;
  Suspects[playerId].currentLocation =
getRoomId(Suspects[playerId].currPosition.x,Suspects[player
Id].currPosition.y);

  printf("%s moved from (%d,%d) to (%d,%d).\n",
Suspects[playerId].name, from->x, from->y, to->x, to->y);

}


void showDisproval( unsigned int sender, unsigned int
receiver, unsigned int cardIndex)
{
//    int i;
  //if receiver = myplayer.id then display card else
display message:sender has shown a card to receiver.
  if( cardIndex == INVALID_CARD_ID){
    if( getNextPlayer(sender) == MyPlayer.Id && Turn !=
MyPlayer.Id)
      disprove();
    return;
  }

    printf("%s disproved with ", Suspects[sender].name);
    writeSpriteToSram( Suspects[sender].spriteAddr +
MY_MSG_OFFSET, ACCUSE_BASEX+0 , ACCUSE_BASEY );
    if(receiver==MyPlayer.Id)   {   // show disprove card
        switch(Cards[cardIndex].cardType) {
            case cardPerson :
                printf("
%s\n",Suspects[Cards[cardIndex].Index].name);
//                for(i=0; i< 3; i++)     {   // blink
```

```
//
 writeSpriteToSram(Suspects[Cards[cardIndex].Index].spriteA
ddr, -1, -1);
//                    writeSpriteToSram(
Suspects[sender].spriteAddr + MY_MSG_OFFSET, -1 , -1 );
//                    writeSpriteToSram( 53 ,
ACCUSE_BASEX+0 , ACCUSE_BASEY );
//                    writeSpriteToSram(55, ACCUSE_BASEX+2
, ACCUSE_BASEY );
//writeSpriteToSram(Suspects[Cards[cardIndex].Index].sprite
Addr + ACCUSE_MSG_OFFSET, ACCUSE_BASEX+2 , ACCUSE_BASEY );

                   writeSpriteToSram(
Suspects[sender].spriteAddr + MY_MSG_OFFSET, ACCUSE_BASEX+0
, ACCUSE_BASEY );

 writeSpriteToSram(BGND_B1,ACCUSE_BASEX+2,ACCUSE_BASEY);
//                }
               break;
            case cardWeapon :
               printf("
%s\n",Weapons[Cards[cardIndex].Index].name);
//                for(i=0; i< 3; i++)     { // blink
//
 writeSpriteToSram(Weapons[Cards[cardIndex].Index].spriteAd
dr, -1, -1);
//                    writeSpriteToSram(
Suspects[sender].spriteAddr + MY_MSG_OFFSET, -1 , -1 );
//                    writeSpriteToSram( 53 ,
ACCUSE_BASEX+0 , ACCUSE_BASEY );
//                    writeSpriteToSram(55, ACCUSE_BASEX+3
, ACCUSE_BASEY );
                   //writeSpriteToSram(
Suspects[sender].spriteAddr + MY_MSG_OFFSET, ACCUSE_BASEX+0
, ACCUSE_BASEY );
                   //writeSpriteToSram(
Weapons[Cards[cardIndex].Index].spriteAddr +
ACCUSE_MSG_OFFSET, ACCUSE_BASEX+3 , ACCUSE_BASEY );
                   writeSpriteToSram(
Suspects[sender].spriteAddr + MY_MSG_OFFSET, ACCUSE_BASEX+0
, ACCUSE_BASEY );

 writeSpriteToSram(BGND_B1,ACCUSE_BASEX+3,ACCUSE_BASEY);
//                }
               break;
            case cardRoom:
               printf("
```

```
%s\n",Rooms[Cards[cardIndex].Index].name);
//                for(i=0; i< 3; i++)     { // blink
  //
 writeSpriteToSram(Rooms[Cards[cardIndex].Index].spriteAddr
, -1, -1);
  //                  writeSpriteToSram(
Suspects[sender].spriteAddr + MY_MSG_OFFSET, -1 , -1 );
  //                  writeSpriteToSram( 53 ,
ACCUSE_BASEX+0 , ACCUSE_BASEY );
  //                  writeSpriteToSram(55, ACCUSE_BASEX+3
, ACCUSE_BASEY );

                    //writeSpriteToSram(
Suspects[sender].spriteAddr + MY_MSG_OFFSET, ACCUSE_BASEX+0
, ACCUSE_BASEY );
                    //writeSpriteToSram(
Rooms[Cards[cardIndex].Index].spriteAddr +
ACCUSE_MSG_OFFSET, ACCUSE_BASEX+4 , ACCUSE_BASEY );
                    writeSpriteToSram(
Suspects[sender].spriteAddr + MY_MSG_OFFSET, ACCUSE_BASEX+0
, ACCUSE_BASEY );

 writeSpriteToSram(BGND_B1,ACCUSE_BASEX+4,ACCUSE_BASEY);
  //              }
               break;
           default :
//                for(i=0; i< 3; i++)     { // blink
    //                writeSpriteToSram(
Suspects[sender].spriteAddr + MY_MSG_OFFSET, -1 , -1 );
    //                  writeSpriteToSram( 53 ,
ACCUSE_BASEX+0 , ACCUSE_BASEY );
                   // writeSpriteToSram(
Suspects[sender].spriteAddr + MY_MSG_OFFSET, ACCUSE_BASEX+0
, ACCUSE_BASEY );
                    writeSpriteToSram(
Suspects[sender].spriteAddr + MY_MSG_OFFSET, ACCUSE_BASEX+0
, ACCUSE_BASEY );

 writeSpriteToSram(BGND_B1,ACCUSE_BASEX+5,ACCUSE_BASEY);
  //              }
               break;
        }
    }
    else    {   // others, just blink sender card
//        for(i=0; i< 3; i++)     { // blink
      //      writeSpriteToSram(
Suspects[sender].spriteAddr + MY_MSG_OFFSET, -1 , -1 );
```

```
      //        writeSpriteToSram( 53 , ACCUSE_BASEX+0 ,
ACCUSE_BASEY );
              //writeSpriteToSram(
Suspects[sender].spriteAddr + MY_MSG_OFFSET, ACCUSE_BASEX+0
, ACCUSE_BASEY );
              writeSpriteToSram(
Suspects[sender].spriteAddr + MY_MSG_OFFSET, ACCUSE_BASEX+0
, ACCUSE_BASEY );

 writeSpriteToSram(BGND_B1,ACCUSE_BASEX+0,ACCUSE_BASEY);

//        }
    }
  if(Turn==MyPlayer.Id){
    invalidateSuggestion();
  }

}

void showSuggestion(unsigned int playerId, unsigned int
suspect, unsigned int weapon, unsigned int room)
{

  //update final position of suspect and weapon AND
currlocation, prevlocation
  move_weapon_to_room(weapon,room);

Suspects[suspect].currPosition=finalPositionInRoom(Rooms[ro
om]);
     Suspects[suspect].currentLocation=room;
     Suspects[suspect].prevVisited=INVALID_ROOM_ID;
     printf("Moved Suspect %d into Room %d at
(%d,%d)\n",suspect,room,Suspects[suspect].currPosition.x,Su
spects[suspect].currPosition.y);
     writeSpriteToSram( Suspects[suspect].spriteAddr +
BD_OFFSET, Suspects[suspect].currPosition.x ,
Suspects[suspect].currPosition.y );

  //change display + change position of weapon + change
position of suspect in suggestion

  PlayerSuggestion[0] = suspect;
  PlayerSuggestion[1] = weapon;
  PlayerSuggestion[2] = room;
  writeSpriteToSram( Suspects[playerId].spriteAddr +
MY_MSG_OFFSET, ACCUSE_BASEX+0 , ACCUSE_BASEY );
```

```
  writeSpriteToSram( SUGGESTION_B, ACCUSE_BASEX+1 ,
ACCUSE_BASEY );
  writeSpriteToSram( Suspects[suspect].spriteAddr +
ACCUSE_MSG_OFFSET, ACCUSE_BASEX+2 , ACCUSE_BASEY );
  writeSpriteToSram( Weapons[weapon].spriteAddr +
ACCUSE_MSG_OFFSET, ACCUSE_BASEX+3 , ACCUSE_BASEY );
  writeSpriteToSram( Rooms[room].spriteAddr +
ACCUSE_MSG_OFFSET, ACCUSE_BASEX+4 , ACCUSE_BASEY );

  printf("%s suggested Suspect: %s, Weapon: %s, Room: %s
\n", Suspects[playerId].name,
Suspects[PlayerSuggestion[0]].name,
Weapons[PlayerSuggestion[1]].name,
Rooms[PlayerSuggestion[2]].name);
}

inline void invalidateSuggestion()
{
  PlayerSuggestion[0] = INVALID_CARD_ID;
  PlayerSuggestion[1] = INVALID_CARD_ID;
  PlayerSuggestion[2] = INVALID_CARD_ID;

  writeSpriteToSram(
Suspects[PlayerSuggestion[0]].spriteAddr +
ACCUSE_MSG_OFFSET, -1, -1 );
  writeSpriteToSram(
Weapons[PlayerSuggestion[1]].spriteAddr +
ACCUSE_MSG_OFFSET, -1, -1 );
  writeSpriteToSram( Rooms[PlayerSuggestion[2]].spriteAddr
+ ACCUSE_MSG_OFFSET, -1, -1 );
}

void sendChangeTurnPacket()
{
  Packet *pkt;

  do {
    if( Turn==PlayerCount-1 )
      Turn =0;
    else
      Turn++;
  }while(!Suspects[Turn].playing);

  pkt = (Packet *)malloc(sizeof(Packet));

  pkt->packetType = ChangeTurn;
  pkt->From = MyPlayer.Id;
```

```
  pkt->To = INVALID_PLAYER_ID;
  pkt->u.s8.senderPlayerId=MyPlayer.Id;
  pkt->u.s8.nextPlayerId = Turn;

printf("change turn from %s to %s\n", Suspects[pkt-
>u.s8.senderPlayerId].name, Suspects[pkt-
>u.s8.nextPlayerId].name);
  sendPacket(pkt);
  invalidateSuggestion();

  free(pkt);
}

void showChangeTurn(unsigned int sender, unsigned int
nextPlayer)
{
  //change display

  printf("Change turn from %s to %s\n",
Suspects[sender].name, Suspects[nextPlayer].name);

  Turn = nextPlayer;
  invalidateSuggestion();

/*  if(nextPlayer==MyPlayer.Id)
    give control to keyboard
    {
         play();

    }
*/
}

void showAccusation( unsigned int playerId, unsigned int
suspect, unsigned int weapon, unsigned int room)
{
  //change display

  // update suspect weapon postion/room location
  move_weapon_to_room(weapon,room);
  move_suspect_to_room(weapon,room);

  PlayerSuggestion[0] = suspect;
  PlayerSuggestion[1] = weapon;
  PlayerSuggestion[2] = room;
  writeSpriteToSram( Suspects[playerId].spriteAddr +
MY_MSG_OFFSET, ACCUSE_BASEX+0 , ACCUSE_BASEY );
```

```
  writeSpriteToSram( SUGGESTION_B, ACCUSE_BASEX+1 ,
ACCUSE_BASEY );
  writeSpriteToSram( Suspects[suspect].spriteAddr +
ACCUSE_MSG_OFFSET, ACCUSE_BASEX+2 , ACCUSE_BASEY );
  writeSpriteToSram( Weapons[weapon].spriteAddr +
ACCUSE_MSG_OFFSET, ACCUSE_BASEX+3 , ACCUSE_BASEY );
  writeSpriteToSram( Rooms[room].spriteAddr +
ACCUSE_MSG_OFFSET, ACCUSE_BASEX+4 , ACCUSE_BASEY );

  printf("Accusation: %s did using %s in %s\n ",
Suspects[suspect].name, Weapons[weapon].name,
Rooms[room].name );

}

void showGameOver(unsigned int winnerPlayerId)
{
  printf("%s is the winner.\n",
Suspects[winnerPlayerId].name);

}

void showPlayerOut(unsigned int playerId)
{

  Suspects[playerId].playing =0;
  printf("%d is out\n",playerId);

}

/*
int main(){
  return 0;
}

*/
#include "basic_io.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> //for usleep()
#include <alt_types.h>
#include "alt_up_ps2_port.h"
#include "ps2_keyboard.h"
#include "netinterface.h"
#include "keyboard.h"
#include "basictypes.h"
```

```
extern void invalidateSuggestion();
//#include ".h"
//#include "ps2_mouse.h"

extern unsigned int getNextPlayer(unsigned int);
#define PLAY_OPTIONS 4
#define DICE 0
#define SECRET_PASSAGE 1
#define SUGGESTION 2
#define ACCUSATION 3

#define SUSPECT 0
#define WEAPON 1
#define ROOM 2

void testMessage(unsigned int index) {

switch(index){
case DICE:
printf("throw dice: press enter\n");
break;
case SECRET_PASSAGE:
printf("secret passage: press enter\n");
break;
case SUGGESTION:
printf("suggestion: press enter\n");
break;
case ACCUSATION:
printf("accusation: press enter\n");
break;

default : break;
}

}
void testWalkMessage() {

printf("left: J; right K; up: I; down: M \n");

}
/*void blinkDisproveSelection(int index)        {
        unsigned int position, data;

        position = 0x00ff;
        data = (Cards[MyPlayer.cards[index]].spriteAddr <<
16)|position;
        IOWR_CONTROLLER_DATA(BASEADDR, data);        //
```

```
reset sprite
//        usleep(BLINK_DELAY);
        writeSpriteToSram(
Cards[MyPlayer.cards[index]].spriteAddr, CARD_BASE + (index
% 5), index /5  );        // set sprite

}
void blinkPlayOptionSelection(int index) {
        unsigned int position, data;

        writeSpriteToSram(BALLROOM_B,
PLAY_OPTION_BASEX+index,PLAY_OPTION_BASEY);// reset sprite
by covering with background tile
//        usleep(BLINK_DELAY);
        position = 0x00ff;
        data = (BALLROOM_B << 16)|position;
        IOWR_CONTROLLER_DATA(BASEADDR, data);// set sprite
by removing background tile
}
void blinkSuggestionSelection(int sId, int wId, int rId)
     {
        unsigned int position, data;

        writeSpriteToSram(BGND_B1,
PLAY_OPTION_BASEX,PLAY_OPTION_BASEY+sId);// reset sprite by
covering with background tile
//        usleep(BLINK_DELAY/4);
        writeSpriteToSram(BGND_B2,
PLAY_OPTION_BASEX+2,PLAY_OPTION_BASEY+wId);// reset sprite
by covering with background tile
//        usleep(BLINK_DELAY/4);
        writeSpriteToSram(BGND_B1,
PLAY_OPTION_BASEX+4,PLAY_OPTION_BASEY+rId);// reset sprite
by covering with background tile
//        usleep(BLINK_DELAY/4);
        position = 0x00ff;
        data = (BALLROOM_B << 16)|position;
        IOWR_CONTROLLER_DATA(BASEADDR, data);// set sprite
by removing background tile
}
*/
int initialize_keyboard()        {
         switch (get_mode()) {
                  case PS2_KEYBOARD:
                           return 1;
                  case PS2_MOUSE:
                           printf("Error: Mouse detected on
```

```
PS/2 port\n");
                            return 0;
                    default:
                            printf("Error: Unrecognized or
no device on PS/2 port\n");
                            return 0;
          }
}

void disprove()         {
          KB_CODE_TYPE decode_mode;
          alt_u8 key=0;
          Packet *pkt;
          int status, i;
      typeIndex index=0;
      writeSpriteToSram(BGND_B1,CARD_BASE+(index%5),(index/5
+1));

          pkt = (Packet *)malloc(sizeof(Packet));
          pkt->From=MyPlayer.Id;

     fflush(stdout);
     reset_keyboard();
     printf("Disprove menu: left card: J; right card:K;
Disprove with selected card: enter\n");
     for(i=0;i<MyPlayer.cardCount;i++)   {
         switch(Cards[MyPlayer.cards[i]].cardType) {
             case cardPerson :
                 printf("Suspect card
%s\n",Suspects[Cards[MyPlayer.cards[i]].Index].name);
                 break;
             case cardWeapon :
                 printf("Weapon card
%s\n",Weapons[Cards[MyPlayer.cards[i]].Index].name);
                 break;
             case cardRoom:
                 printf("Room card
%s\n",Rooms[Cards[MyPlayer.cards[i]].Index].name);
                 break;
             default : break;
         }
     }
          for(;;)         {
         switch(Cards[MyPlayer.cards[index]].cardType) {
             case cardPerson :
                 printf("Suspect card
%s\n",Suspects[Cards[MyPlayer.cards[index]].Index].name);
```

```
                break;
            case cardWeapon :
                printf("Weapon card
%s\n",Weapons[Cards[MyPlayer.cards[index]].Index].name);
                break;
            case cardRoom:
                printf("Room card
%s\n",Rooms[Cards[MyPlayer.cards[index]].Index].name);
                break;
            default : break;
        }
          fflush(stdout);
                  status=read_make_code(&decode_mode,&key);
                  if(status==PS2_SUCCESS)        {
                           switch(decode_mode)        {
                                        case
KB_BINARY_MAKE_CODE :
                                                    switch
(key) {

 case  0x5a: //enter key

        if(validate_card(index))        {

                     pkt->packetType=Disprove;

                     pkt->To=Turn;

                     pkt->u.s7.senderPlayerId=MyPlayer.Id;

                     pkt->u.s7.receiverPlayerId=Turn;

                     if(index==MyPlayer.cardCount)        {
// cannot disprove suggestion, send INVALID_CARD_ID

                                 pkt-
>u.s7.cardIndex=INVALID_CARD_ID;

DisproveTurn=getNextPlayer(MyPlayer.Id);
                                        }

                     else        { // send valid card

                                 pkt-
>u.s7.cardIndex=MyPlayer.cards[index];

DisproveTurn=INVALID_CARD_ID;
```

```
                                        }

                    printf("Player %d disproves %d card
%d\n",MyPlayer.Id,Turn,MyPlayer.cards[index]);

                    sendPacket(pkt);

                    // disprove_card_send_video();

                    return;

        }

        break;
                          default :break;
                        }
                    case KB_ASCII_MAKE_CODE :
                        switch (key) {


 case 0x4a: // J 0x6b:  // Left arrow

        if(index!=0)        {

                    index=index-1;

 writeSpriteToSram(BGND_B1,CARD_BASE+(index%5),(index/5
+1));

                    // highlight_video(index);


        }

        break;

 case 0x4b: // K 0x74:  // Right arrow

        if(index<MyPlayer.cardCount) {

                    index=index+1;

 writeSpriteToSram(BGND_B1,CARD_BASE+(index%5),(index/5
+1));

                    // highlight_video(index);
```

```
        }

        break;
                                                    }
                                            break;
                                            default : break;
                                }
                        }
         }
         free(pkt);
}

void play()        {
         KB_CODE_TYPE decode_mode;
         alt_u8 key=0;
         Packet *pkt;
     typeIndex index=0;
         int mode=-1, status, diceNum;
     typeIndex accCards[CONFIDENTIAL_FILE_SIZE];

     writeSpriteToSram(BGND_B1,
PLAY_OPTION_BASEX+index,PLAY_OPTION_BASEY);

         pkt = (Packet *)malloc(sizeof(Packet));
         pkt->From=MyPlayer.Id;

         /** Pick one of the 3 playing modes
          * Roll dice and move,
          * Take secret passage
          * Make suggestion if already in a room
          **/

 fflush(stdout);
 reset_keyboard();
printf("start gameplay!\n");
testMessage(index);
         for(;;)        {
        fflush(stdout);
                  status=read_make_code(&decode_mode,&key);
                  if(status==PS2_SUCCESS)        {
                           switch(decode_mode)        {
                                    case
KB_BINARY_MAKE_CODE :
                                                switch
(key) {

 case  0x5a: //enter key
```

```
        if(validate_mode(index))

                    mode=index;


        break;
                        }
                     break;
                   case KB_ASCII_MAKE_CODE :
printf("ascii %x \n", key);
                       switch(key){

 case 0x4a : // J 0x6b:  // Left arrow

        if(index!=0)        {

                    index=index-1;

 writeSpriteToSram(BGND_B1,
PLAY_OPTION_BASEX+index,PLAY_OPTION_BASEY);

                    // highlight_video();

        }
testMessage(index);

        break;

 case 0x4b : //K 0x74:  // Right arrow

        if(index+1<PLAY_OPTIONS) {

                    index=index+1;

 writeSpriteToSram(BGND_B1,
PLAY_OPTION_BASEX+index,PLAY_OPTION_BASEY);

                    // highlight_video();

        }
testMessage(index);

        break;
                                                    }
                        break;
```

```
                default : break;
               }
                  }
                  if(mode!=-1)
                           break;
         }
         switch(mode)        {
                  case DICE :
                           diceNum = getRandom(2,12);
                           pkt->packetType=DiceRolled;
                           pkt->To=INVALID_PLAYER_ID;
                           pkt->u.s4.value=diceNum;
                           pkt->u.s4.playerId=MyPlayer.Id;
                           sendPacket(pkt);
               outport(SEG7_DISPLAY_BASE, (unsigned
int)diceNum);
                           // display_video(diceNum);
printf("you can move %d places\n", diceNum);
                           walk_pathway(diceNum);
                           if(player_inside_room())
                                       make_suggestion();
                           else

 sendChangeTurnPacket();
                           return;
                  case SECRET_PASSAGE :

Suspects[MyPlayer.Id].prevVisited=Suspects[MyPlayer.Id].cur
rentLocation;

Suspects[MyPlayer.Id].currentLocation=Rooms[Suspects[MyPlay
er.Id].currentLocation].secretPassage;

Suspects[MyPlayer.Id].nextPosition=finalPositionInRoom(Room
s[Suspects[MyPlayer.Id].currentLocation]);

                           pkt->packetType=FinalPosition;
     // Move packet
                           pkt->To=INVALID_PLAYER_ID;
                           pkt->u.s5.playerId=MyPlayer.Id;
                            pkt-
>u.s5.oldPositionX=Suspects[MyPlayer.Id].currPosition.x;
                           pkt-
>u.s5.oldPositionY=Suspects[MyPlayer.Id].currPosition.y;
                           pkt-
>u.s5.newPositionX=Suspects[MyPlayer.Id].nextPosition.x;
                           pkt-
```

```
>u.s5.newPositionY=Suspects[MyPlayer.Id].nextPosition.y;

                            sendPacket(pkt);
                            // display_video();
                writeSpriteToSram(
Suspects[MyPlayer.Id].spriteAddr + BD_OFFSET,
Suspects[MyPlayer.Id].nextPosition.x,
Suspects[MyPlayer.Id].nextPosition.y );
                            make_suggestion();

Suspects[MyPlayer.Id].currPosition=Suspects[MyPlayer.Id].ne
xtPosition;
                            return;
                    case SUGGESTION :
                            make_suggestion();
                            return;
                    case ACCUSATION:
                            make_accusation(accCards);

if(validate_accusation(accCards))        {
                                        pkt-
>packetType=GameOver;
                                        pkt-
>To=INVALID_PLAYER_ID;
                                        pkt-
>u.s10.winnerId=MyPlayer.Id;
                                        sendPacket(pkt);
                                        printf("Game over!!
You won!!\n");
                                        // display()
                                        // exit game?
                            }
                            else {

 Suspects[MyPlayer.Id].playing=0;
                                        pkt-
>packetType=PlayerOut;
                                        pkt-
>To=INVALID_PLAYER_ID;
                                        pkt-
>u.s11.playerId=MyPlayer.Id;
                                        sendPacket(pkt);
                                        // display
confidential cards

 show_confidential_cards();
                                        // display "not
```

```
allowed to play anymore"
                                        printf("You are not
allowed to play further\n");
                                        printf("You must
stay in the game and disprove suggestions\n");

 sendChangeTurnPacket();
                            }

                        return;
                default : break;
         }

         free(pkt);
}

int validate_card(typeIndex ind)        {
         int j;

         if(ind==MyPlayer.cardCount)        {        // No
card to show
        printf("No card to show, check this\n");
        for(j=0;j<MyPlayer.cardCount;j++)        {// Check
whether really no card to show
            switch(Cards[MyPlayer.cards[j]].cardType)   {
                case cardPerson:

 if(Cards[MyPlayer.cards[j]].Index==PlayerSuggestion[0]) {
                        printf("You have a suggestion
card\n");
                         // display invalid card msg
                         return 0;  // Invalid
                    }
                    break;
                case cardWeapon:

 if(Cards[MyPlayer.cards[j]].Index==PlayerSuggestion[1]) {
                        printf("You have a suggestion
card\n");
                         // display invalid card msg
                         return 0;  // Invalid
                    }
                    break;
                case cardRoom:

 if(Cards[MyPlayer.cards[j]].Index==PlayerSuggestion[2]) {
                        printf("You have a suggestion
```

```
card\n");
                        // display invalid card msg
                        return 0;  // Invalid
                    }
                    break;
                default:
                    break;
                        }
                }
                return 1;
        }
        else        {                // valid card, then
check if it is part of the suggestion


 switch(Cards[MyPlayer.cards[ind]].cardType)   {
                case cardPerson:

 if(Cards[MyPlayer.cards[ind]].Index==PlayerSuggestion[0])
  {
                        printf("Check disprove card %s
\n",Suspects[Cards[MyPlayer.cards[ind]].Index].name);
                        return 1;
                    }
                    break;
                case cardWeapon:

 if(Cards[MyPlayer.cards[ind]].Index==PlayerSuggestion[1])
  {
                        printf("Check disprove card %s
\n",Weapons[Cards[MyPlayer.cards[ind]].Index].name);
                        return 1;
                    }
                    break;
                case cardRoom:

 if(Cards[MyPlayer.cards[ind]].Index==PlayerSuggestion[2])
  {
                        printf("Check disprove card %s
\n",Rooms[Cards[MyPlayer.cards[ind]].Index].name);
                        return 1;
                    }
                    break;
                default:
                    break;
          }
                  printf("Invalid card\n");
```

```
                // display invalid card msg
                return 0;
        }
        return 0;       // dummy
}

int validate_mode(typeIndex mode)       {
        typeIndex loc, prevLoc;
        switch(mode)       {
                case DICE :
                        return 1;
                case SECRET_PASSAGE :

loc=Suspects[MyPlayer.Id].currentLocation;
                        if(loc==INVALID_ROOM_ID)
 // not currently in a room
                                        return 0;
                        else
if(Rooms[loc].secretPassage==INVALID_ROOM_ID) // in a room
but room does not have secret passage
                                        return 0;
                        else        // room has secret
passage
                                        return 1;
                case SUGGESTION :

loc=Suspects[MyPlayer.Id].currentLocation;

prevLoc=Suspects[MyPlayer.Id].prevVisited;
                        if(prevLoc==INVALID_ROOM_ID)
                                        return 1;
                        else if(loc==prevLoc)
                                        return 0;
                        else
                                        return 1;
                        break;
                default :
                        break;
        }
        return 0;
}

int player_inside_room()       {

if(Suspects[MyPlayer.Id].currentLocation==INVALID_ROOM_ID)
                return 0;
        else
```

```
                return 1;
        return 0;        // dummy return statement
}

void make_suggestion()        {
        KB_CODE_TYPE decode_mode;
        alt_u8 key=0;
        Packet *pkt;
        int status;
     typeIndex index=0, sId=0, wId=0, rId=0;


writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+0,SUGGESTION_BAS
EY+sId);

        pkt = (Packet *)malloc(sizeof(Packet));
        pkt->From=MyPlayer.Id;
        rId=Suspects[MyPlayer.Id].currentLocation;

        /* display_video(sId);
           display_video(wId);
           display_video(rId);
         */
      printf("Make suggestion menu: weapon: J; person: K;
select: enter\n " );

        for(;;)        {
       fflush(stdout);
       printf("Suspect %d %s, ", sId, Suspects[sId].name);
       printf("Weapon %d %s, ", wId, Weapons[wId].name);
       printf("Room %d %s\n", rId, Rooms[rId].name);

 //blinkSuggestionSelection(Suspects[sId].spriteAddr,
Weapons[wId].spriteAddr, Rooms[rId].spriteAddr);
       status=read_make_code(&decode_mode,&key);
                if(status==PS2_SUCCESS)        {
                        switch(decode_mode)        {
                                        case
KB_BINARY_MAKE_CODE :
                                                switch
(key) {

 case  0x5a: //enter key

       // display - move suspect and weapon to room

       move_weapon_to_room(wId,rId);
```

```
        move_suspect_to_room(sId,rId);


        pkt->packetType=Suggestion;

        pkt->To=INVALID_PLAYER_ID;

        pkt->u.s6.playerId=MyPlayer.Id;

        pkt->u.s6.suspectId=sId;

        pkt->u.s6.weaponId=wId;

        pkt->u.s6.roomId=rId;

        pkt->u.s6.suspectX=Suspects[sId].currPosition.x;

        pkt->u.s6.suspectY=Suspects[sId].currPosition.y;

        pkt->u.s6.weaponX=Weapons[wId].position.x;

        pkt->u.s6.weaponY=Weapons[wId].position.y;

        sendPacket(pkt);
                                    DisproveTurn =
getNextPlayer(MyPlayer.Id);

        printf("Player %d Suggestion S %s W %s R
%s\n",MyPlayer.Id,Suspects[sId].name,Weapons[wId].name,Room
s[rId].name);
                                    free(pkt);

Suspects[MyPlayer.Id].prevVisited=Suspects[MyPlayer.Id].cur
rentLocation;
                                    //DISPLAY
                    writeSpriteToSram(
Suspects[MyPlayer.Id].spriteAddr + MY_MSG_OFFSET,
ACCUSE_BASEX+0 , ACCUSE_BASEY );
                    writeSpriteToSram( SUGGESTION_B,
ACCUSE_BASEX+1 , ACCUSE_BASEY );
                                    writeSpriteToSram(
Suspects[sId].spriteAddr + ACCUSE_MSG_OFFSET,
ACCUSE_BASEX+2 , ACCUSE_BASEY );
                                    writeSpriteToSram(
Weapons[wId].spriteAddr + ACCUSE_MSG_OFFSET, ACCUSE_BASEX+3
, ACCUSE_BASEY );
```

```
                                    writeSpriteToSram(
Rooms[rId].spriteAddr + ACCUSE_MSG_OFFSET, ACCUSE_BASEX+4 ,
ACCUSE_BASEY );
                                    return;
                            }
                            break;
                            case KB_ASCII_MAKE_CODE :
                               switch(key){
                                                    case
0x4a:  // Left arrow

 case 0x4b:  // Right arrow

        index=1-index;
                                    if(index==0)

 writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+0,SUGGESTION_BA
SEY+sId);
                                    else

 writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+2,SUGGESTION_BA
SEY+wId);

        // highlight_video();

        break;

 case 0x49:        // Up arrow

        switch(index)        {

                      case SUSPECT:

                               if(sId+1<SUSPECT_MAX)
 {
                                                    sId++;

 writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+0,SUGGESTION_BA
SEY+sId);
                                                    //
display_video(sId);

                                         // highlight_video()

                               }

                               break;
```

```
                        case WEAPON:

                                if(wId+1<WEAPON_MAX)        {
                                                        wId++;

 writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+2,SUGGESTION_BA
SEY+wId);



                                                //
display_video(wId);

                                }

                                break;

                        default:

                                break;

        }

        break;

 case 0x4d:    // Down arrow

        switch(index)        {

                        case SUSPECT:

                                if(sId!=0)        {
                                                  sId--;

 writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+0,SUGGESTION_BA
SEY+sId);



                                                //
display_video(sId);

                                }

                                break;
```

```
                        case WEAPON:

                                if(wId!=0)        {

                                        wId--;


 writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+2,SUGGESTION_BA
SEY+wId);
                                                          //
display_video(wId);

                                }

                                break;

                        default:

                                break;

        }
                                     break;
                                }
                               break;
                         default : break;
                                          }
                        }
         }
         free(pkt);

Suspects[MyPlayer.Id].prevVisited=Suspects[MyPlayer.Id].cur
rentLocation;
}

void make_accusation(typeIndex accCards[])        {
         KB_CODE_TYPE decode_mode;
         alt_u8 key=0;
         Packet *pkt;
         int index=0, sId=0, wId=0, rId, status;


writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+0,SUGGESTION_BAS
EY+sId);

         pkt = (Packet *)malloc(sizeof(Packet));
         pkt->From=MyPlayer.Id;
```

```
         /* display_video(sId);
            display_video(wId);
            display_video(rId);
          */
         for(;;)        {
printf("Make Accusation menu: weapon: J; person: K; room: K
select: enter\n " );
        fflush(stdout);

 //blinkSuggestionSelection(Suspects[sId].spriteAddr,
Weapons[wId].spriteAddr, Rooms[rId].spriteAddr);
                  status=read_make_code(&decode_mode,&key);
                  if(status==PS2_SUCCESS)        {
                           switch(decode_mode)        {
                                       case
KB_BINARY_MAKE_CODE :
                                                  switch
(key) {

 case  0x5a: //enter key

        // display - move suspect and weapon to room

        move_weapon_to_room(wId,rId);

        move_suspect_to_room(sId,rId);


        pkt->packetType=Accusation;

        pkt->To=INVALID_PLAYER_ID;

        pkt->u.s9.playerId=MyPlayer.Id;

        pkt->u.s9.suspectId=sId;

        pkt->u.s9.weaponId=wId;

        pkt->u.s9.roomId=rId;

        pkt->u.s6.suspectX=Suspects[sId].currPosition.x;

        pkt->u.s6.suspectY=Suspects[sId].currPosition.y;

        pkt->u.s6.weaponX=Weapons[wId].position.x;

        pkt->u.s6.weaponY=Weapons[wId].position.y;
```

```
        sendPacket(pkt);

        printf("Player %d Accusation S %d W %d R
%d\n",MyPlayer.Id,sId,wId,rId);
                                    accCards[0]=sId;
                                    accCards[1]=wId;
                                    accCards[2]=rId;
                                    PlayerSuggestion[0]=sId;
                                    PlayerSuggestion[1]=wId;
                                    PlayerSuggestion[2]=rId;

        // display - move suspect and weapon to room

        // display - move suspect and weapon to room

        //Suspects[sId].currentLocation=rId;

        //Suspects[MyPlayer.Id].prevVisited=rId;
                                    free(pkt);

Suspects[MyPlayer.Id].prevVisited=Suspects[MyPlayer.Id].cur
rentLocation;
                                    //DISPLAY
                                    writeSpriteToSram(
Suspects[MyPlayer.Id].spriteAddr + MY_MSG_OFFSET,
ACCUSE_BASEX+0 , ACCUSE_BASEY );
                    writeSpriteToSram( ACCUSATION_B,
ACCUSE_BASEX+1 , ACCUSE_BASEY );
                    writeSpriteToSram(
Suspects[sId].spriteAddr + ACCUSE_MSG_OFFSET,
ACCUSE_BASEX+2 , ACCUSE_BASEY );
                                    writeSpriteToSram(
Weapons[wId].spriteAddr + ACCUSE_MSG_OFFSET, ACCUSE_BASEX+3
, ACCUSE_BASEY );
                                    writeSpriteToSram(
Rooms[rId].spriteAddr + ACCUSE_MSG_OFFSET, ACCUSE_BASEX+4 ,
ACCUSE_BASEY );


        return;
                        }
                        case KB_ASCII_MAKE_CODE :
                           switch(key){

 case 0x6b:  // Left arrow
```

```
        if(index!=0)        {

                    index=(index-1)%(SUGGESTION_SIZE);
                                        switch(index)    {
                                        case SUSPECT:

writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+0,SUGGESTION_BAS
EY+sId);
                                                          //
display_video(sId);
                                             break;
                                        case WEAPON:

writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+2,SUGGESTION_BAS
EY+wId);
                                                          //
display_video(wId);
                                             break;
                                        case ROOM:

writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+4,SUGGESTION_BAS
EY+rId);
                                                          //
display_video(rId);
                                             break;
                                        default:
                                             break;
                                        }


                    // highlight_video();

        }

        break;

 case 0x74:  // Right arrow

        if(index+1<SUGGESTION_SIZE) {

                    index=(index+1)%(SUGGESTION_SIZE);
                                        switch(index)    {
                                        case SUSPECT:

writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+0,SUGGESTION_BAS
EY+sId);
                                                          //
```

```
display_video(sId);
                                                  break;
                                             case WEAPON:

writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+2,SUGGESTION_BAS
EY+wId);
                                                        //
display_video(wId);
                                                  break;
                                             case ROOM:

writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+4,SUGGESTION_BAS
EY+rId);
                                                        //
display_video(rId);
                                                  break;
                                             default:
                                                  break;
                                             }

                        // highlight_video();

        }

        break;

 case 0x75:        // Up arrow

        break;

 case 0x72:    // Down arrow

        switch(index)        {

                        case SUSPECT:

                                  if(sId!=0)        {

                                            sId--;

 writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+0,SUGGESTION_BA
SEY+sId);

                                            //
display_video(sId);

                                  }
```

```
                                break;

                        case WEAPON:

                                if(wId!=0)        {

                                        wId--;

 writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+2,SUGGESTION_BA
SEY+wId);

                                        //
display_video(wId);

                                }

                                break;

                        case ROOM:

                                if(rId!=0)        {

                                        rId--;

 writeSpriteToSram(BGND_B1,SUGGESTION_BASEX+4,SUGGESTION_BA
SEY+rId);

                                        //
display_video(rId);

                                }

                                break;

                        default:

                                break;

        }
                                     break;
                                                    }
                                             break;
                                              default :
break;
                                }
                  }
```

```
        }
        free(pkt);

Suspects[MyPlayer.Id].prevVisited=Suspects[MyPlayer.Id].cur
rentLocation;
}

void show_confidential_cards()        {
        int i;
        Card c;
     invalidateSuggestion();
        for(i=0;i<CONFIDENTIAL_FILE_SIZE;i++)        {
                c=Cards[ConfidentialFile[i]];
                switch(c.cardType)        {
                        case cardPerson:
                                printf("Suspect %s
",Suspects[c.Index].name);
                    writeSpriteToSram(
Suspects[c.Index].spriteAddr + ACCUSE_MSG_OFFSET,
ACCUSE_BASEX+0 , ACCUSE_BASEY );
                                break;
                        case cardWeapon:
                                printf("Weapon %s
",Weapons[c.Index].name);
                    writeSpriteToSram(
Weapons[c.Index].spriteAddr + ACCUSE_MSG_OFFSET,
ACCUSE_BASEX+1 , ACCUSE_BASEY );
                                break;
                        case cardRoom:
                                printf("Room %s
",Rooms[c.Index].name);
                    writeSpriteToSram(
Rooms[c.Index].spriteAddr + ACCUSE_MSG_OFFSET,
ACCUSE_BASEX+2 , ACCUSE_BASEY );
                                break;
                        default:
                                break;
                }
        }
        printf("\n");



}

void walk_pathway(int diceNum) {
     KB_CODE_TYPE decode_mode;
```

```
     alt_u8 key=0;
         Packet *pkt;
         Position path[diceNum+1];        // store current
position
         Position temp;
         int index, status, i;

         pkt = (Packet *)malloc(sizeof(Packet));
         pkt->From=MyPlayer.Id;

         index=0;
         if(player_inside_room()) {        // if player is
inside a room move him to the door tile

 path[index]=Rooms[Suspects[MyPlayer.Id].currentLocation].d
oor;
          pkt->packetType=Move;
          pkt->To=INVALID_PLAYER_ID;
          pkt->u.s5.playerId=MyPlayer.Id;
          pkt-
>u.s5.oldPositionX=Suspects[MyPlayer.Id].currPosition.x;
          pkt-
>u.s5.oldPositionY=Suspects[MyPlayer.Id].currPosition.y;
          pkt->u.s5.newPositionX=path[index].x;
          pkt->u.s5.newPositionY=path[index].y;
          sendPacket(pkt);
          printf("Door position %d
%d\n",path[index].x,path[index].y);
          // display_animation(path[index-1],path[index);
          writeSpriteToSram(
Suspects[MyPlayer.Id].spriteAddr + BD_OFFSET, path[index].x
, path[index].y );
     }
         else

 path[index]=Suspects[MyPlayer.Id].currPosition;



         index++;
         while(index<=diceNum)        {
printf(" %s at %d,%d",Suspects[MyPlayer.Id].name,
Suspects[MyPlayer.Id].currPosition.x,Suspects[MyPlayer.Id].
currPosition.y);
testWalkMessage();
        fflush(stdout);
                  status=read_make_code(&decode_mode,&key);
```

```
                if(status==PS2_SUCCESS)         {
                        switch(decode_mode)         {
                  case KB_ASCII_MAKE_CODE :
                                                  switch
(key) {

 case 0x4a : //0x6b:  // Left arrow

        temp.x=path[index-1].x-1;

        temp.y=path[index-1].y;

        break;

 case 0x4b : //0x74:  // Right arrow

        temp.x=path[index-1].x+1;

        temp.y=path[index-1].y;

        break;

 case 0x49 : //0x75:        // Up arrow

        temp.x=path[index-1].x;

        temp.y=path[index-1].y-1;

        break;

 case 0x4d : //0x72:    // Down arrow

        temp.x=path[index-1].x;

        temp.y=path[index-1].y+1;

        break;
                          }
                  break;
                  case KB_BINARY_MAKE_CODE :
                      switch(key){

 case  0x5a: //enter key

 // stop walking and return

          if(!player_inside_room())        // if player is
```

```
blocked, do not set his position to door tile


 Suspects[MyPlayer.Id].currPosition=path[index-1];

        return;

                        }
                        break;
                                    default:
                    continue;
                                     temp.x=path[index-
1].x;
                                   temp.y=path[index-1].y;
                                   break;
                                 }

if(validate_position(temp,path,index))        {

 path[index]=temp;

 pkt->packetType=Move;

 pkt->To=INVALID_PLAYER_ID;

 pkt->u.s5.playerId=MyPlayer.Id;

 pkt->u.s5.oldPositionX=path[index-1].x;

 pkt->u.s5.oldPositionY=path[index-1].y;

 pkt->u.s5.newPositionX=path[index].x;

 pkt->u.s5.newPositionY=path[index].y;

 sendPacket(pkt);

 printf("Old position %d %d\n",path[index-1].x,path[index-
1].y);

 printf("New position %d
%d\n",path[index].x,path[index].y);

 // display_animation(path[index-1],path[index);

 writeSpriteToSram( Suspects[MyPlayer.Id].spriteAddr +
BD_OFFSET, path[index].x , path[index].y );
```

```
 for(i=0;i<ROOM_COUNT;i++)        {        // entered room
through door?


if((Rooms[i].door.x==path[index].x)&&(Rooms[i].door.y==path
[index].y))        {

                      pkt->packetType=FinalPosition;

                      pkt->To=INVALID_PLAYER_ID;

                      pkt->u.s5.playerId=MyPlayer.Id;

                      pkt->u.s5.oldPositionX=path[index].x;

                      pkt->u.s5.oldPositionY=path[index].y;

                      // To do calculate final position in
room

                      temp=finalPositionInRoom(Rooms[i]);

                      pkt->u.s5.newPositionX=temp.x;

                      pkt->u.s5.newPositionY=temp.y;

                      sendPacket(pkt);

                      printf("Valid door tile\n");

                      writeSpriteToSram(
Suspects[MyPlayer.Id].spriteAddr + BD_OFFSET, temp.x,
temp.y );

                      // SMOOTH MARKER


 Suspects[MyPlayer.Id].currPosition=temp;


 Suspects[MyPlayer.Id].currentLocation=i;
```

```
 Suspects[MyPlayer.Id].prevVisited=INVALID_ROOM_ID;

                    return;

        }
                                                                  }

 index++;
                                          }
                   }
      }
         Suspects[MyPlayer.Id].currPosition=path[index-1];
               // ended up on pathway tile

Suspects[MyPlayer.Id].currentLocation=INVALID_ROOM_ID;
         Suspects[MyPlayer.Id].prevVisited=INVALID_ROOM_ID;
         free(pkt);
}

int validate_position(Position pos, Position path[], int
index)        {
         int i;
         Tile t;


if(pos.x<0||pos.x>=COL_MAX||pos.y<0||pos.y>=ROW_MAX)
 { // spilling out of game board
                  printf("Spilling out of game board\n");
                  return 0;
         }
                  // miss turn if inside room and someone
blocking the door
                  // if last tile is a pathway tile make
prevLocation of player INVALID_ROOM_ID
         t=GameBoard[pos.y][pos.x];
         switch(t.tileType)        {
                  case tileInvalid:        // some center
tiles of board are invalid
                           return 0;
                  case tileRoom:        // invalid if tile
is not a door
                           for(i=0;i<ROOM_COUNT;i++)
 {

 if((Rooms[i].door.x==pos.x)&&(Rooms[i].door.y==pos.y))
   {
```

```
printf("Valid door tile\n");
                                                    return 1;
                                            }
                            }
                            printf("Invalid room tile\n");
                            return 0;
                    case tilePathway:
                            i=0;

Suspects[MyPlayer.Id].prevVisited=INVALID_ROOM_ID;
                            while(i<index)        {

 if((pos.x==path[i].x)&&(pos.y==path[i].y))        {
 // reached earlier tile in path

printf("Reached earlier tile in path\n");
                                                    return 0;
                                            }
                                            i++;
                            }
                            for(i=0;i<SUSPECT_MAX;i++)
 {        // Not allowed to walk over another pawn

 if((pos.x==Suspects[i].currPosition.x)&&(pos.y==Suspects[i
].currPosition.y))        {

printf("Not allowed to walk over another pawn\n");
                                                    return 0;
                                            }
                            }
                            printf("Valid pathway tile\n");
                            return 1;
                    default:
                            break;
            }
            return 0;        // dummy
}

Position finalPositionInRoom(Room room)        {
            Position pos;
            int x,i;

            pos.x=-1; pos.y=-1;
            // (topLeft.x, topLeft.y) is place for weapons
            for(x=room.topLeft.x+2;x<room.bottomRight.x;x=x+2)
          {
                    for(i=0;i<SUSPECT_MAX;i++)        {
```

```
                        if(x==Suspects[i].currPosition.x
&& room.topLeft.y + 1==Suspects[i].currPosition.y)        {
                                    pos.x=-1;
                                    pos.y=-1;
                                    break;
                        }
                        else        {
                                    pos.x=x;

 pos.y=room.topLeft.y + 1;
                        }
                }
                if(pos.x!=-1&&pos.y!=-1)        // no
pawn on this position
                        return pos;
        }
        for(x=room.topLeft.x;x<room.bottomRight.x;x=x+2)
    {
                for(i=0;i<SUSPECT_MAX;i++)        {
                        if(x==Suspects[i].currPosition.x
&& room.bottomRight.y==Suspects[i].currPosition.y)        {
                                    pos.x=-1;
                                    pos.y=-1;
                                    break;
                        }
                        else        {
                                    pos.x=x;

 pos.y=room.bottomRight.y ;
                        }
                }
                if(pos.x!=-1&&pos.y!=-1)        // no
pawn on this position
                        return pos;
        }
        return pos;
}

void move_weapon_to_room(typeIndex wId, typeIndex rId)
   {
        int i;

        for(i=0;i<WEAPON_MAX;i++)        {
                if(i!=wId)        {
                        if(rId==Weapons[i].roomIndex)
    {
```

```
 Weapons[i].roomIndex=Weapons[wId].roomIndex;

 Weapons[wId].roomIndex=rId;

 Weapons[i].position.x=Rooms[Weapons[i].roomIndex].topLeft.
x;

 Weapons[i].position.y=Rooms[Weapons[i].roomIndex].topLeft.
y +1;

 Weapons[wId].position.x=Rooms[Weapons[wId].roomIndex].topL
eft.x;

 Weapons[wId].position.y=Rooms[Weapons[wId].roomIndex].topL
eft.y +1;
                                        printf("Moved
Weapon %d into Room %d\n",i,Weapons[i].roomIndex);
                                        printf("Moved
Weapon %d into Room %d\n",wId,rId);
                                        writeSpriteToSram(
Weapons[i].spriteAddr + BD_OFFSET, Weapons[i].position.x ,
Weapons[i].position.y );
                                        writeSpriteToSram(
Weapons[wId].spriteAddr + BD_OFFSET,
Weapons[wId].position.x , Weapons[wId].position.y );

                                        return;
                            }
                    }
          }
          Weapons[wId].roomIndex=rId;

Weapons[wId].position.x=Rooms[Weapons[wId].roomIndex].topLe
ft.x;

Weapons[wId].position.y=Rooms[Weapons[wId].roomIndex].topLe
ft.y +1;
          writeSpriteToSram( Weapons[wId].spriteAddr +
BD_OFFSET, Weapons[wId].position.x ,
Weapons[wId].position.y );

}

void move_suspect_to_room(typeIndex sId, typeIndex rId)
   {
  if( sId != MyPlayer.Id){
```

```
Suspects[sId].currPosition=finalPositionInRoom(Rooms[rId]);
         Suspects[sId].currentLocation=rId;
         Suspects[sId].prevVisited=INVALID_ROOM_ID;
         printf("Moved Suspect %d into Room %d at
(%d,%d)\n",sId,rId,Suspects[sId].currPosition.x,Suspects[sI
d].currPosition.y);
         writeSpriteToSram( Suspects[sId].spriteAddr +
BD_OFFSET, Suspects[sId].currPosition.x ,
Suspects[sId].currPosition.y );
  }
}

int validate_accusation(typeIndex accCards[]) {
    int i, j, found;

    for(i=0;i<CONFIDENTIAL_FILE_SIZE;i++)   {
        found=0;
        for(j=0;j<CONFIDENTIAL_FILE_SIZE;j++)   {
            if(accCards[i]==ConfidentialFile[j])    {
                found=1;
                break;
            }
        }
        if(!found)  // card not found in ConfidentialFile
            return 0;
    }
    return 1;   // all cards match confidential file
}

#include "basic_io.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alt_types.h>
#include "alt_up_ps2_port.h"
#include "DM9000A.h"

#include "keyboard.h"
#include "netinterface.h"
#include "gamelogic.h"

#define MAX_MSG_LENGTH 128
#define MAX_BUFFER_SIZE 1600

static unsigned char mac_address[6] = { 0x01, 0x60,  0x6E,
0xbb, 0xbb, 0xbb  };
```

```
static unsigned int receive_buffer_length;
static unsigned char receive_buffer[MAX_BUFFER_SIZE];

static unsigned int interrupt_number;
static unsigned int IP_count =0;
static int curMsgChar = 0;


#define UDP_PACKET_PAYLOAD_OFFSET 42
#define UDP_PACKET_LENGTH_OFFSET 38
#define IP_ID_OFFSET 18
#define IP_CHECKSUM_OFFSET 24
#define UDP_CHECKSUM_OFFSET 40

#define UDP_PACKET_PAYLOAD (transmit_buffer +
UDP_PACKET_PAYLOAD_OFFSET)

static unsigned char transmit_buffer[] = {
  // Ethernet MAC header
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, // Destination MAC
address
  0x01, 0x60, 0x6E, 0x11, 0x02, 0x0F, // Source MAC address
  0x08, 0x00,                         // Packet Type: 0x800
= IP

  // IP Header
  0x45,               // version (IPv4), header length =
20 bytes
  0x00,               // differentiated services field
  0x00,0x9C,          // total length: 20 bytes for IP
header +
                      // 8 bytes for UDP header + 128
bytes for payload
  0x00, 0x00,         // packet ID   18 and 19
  0x00,               // flags
  0x00,               // fragment offset
  0x80,               // time-to-live
  0x11,               // protocol: 11 = UDP
  0x00,0x00,          // header checksum: incorrect  24 25
  0xc0,0xa8,0x01,0x01, // source IP address
  0xc0,0xa8,0x01,0xff, // destination IP address

  // UDP Header
  0x67,0xd9, // source port port (26585: garbage)  34 35
  0x27,0x2b, // destination port (10027: garbage)  36 37
  0x00,0x88, // length (72: 8 for UDP header + 64 for data)
38 39
```

```
  0x00,0x00, // checksum: 0 = none  40 41

  // UDP payload
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
    0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67

};

void printPacket(unsigned char *pkt, unsigned int len)
{
    printf("####");
    int i=0;
    for(i=0; i<len;++i){
        printf("%x ", pkt[i] );
    }
    printf("####");

}
void networkInitialize()
{
  printf("net Init\n");
  int i;
  int interrupt_number;

  printf("net Init2\n");

  // Initalize the DM9000 and the Ethernet interrupt
handler
  DM9000_init(mac_address);
  interrupt_number = 0;
  alt_irq_register(DM9000A_IRQ, NULL,
(void*)ethernet_interrupt_handler);
```

```
  for( i =0 ; i<6;++i) {
   transmit_buffer[i+6]= mac_address[i];
  }

  // Clear the payload
  for (curMsgChar=MAX_MSG_LENGTH-1; curMsgChar>0;
curMsgChar--) {
    UDP_PACKET_PAYLOAD[curMsgChar] = 0;
  }

}

unsigned int checksum( int start, int end, int
PACKET_OFFSET) {

unsigned short word16;
unsigned int sum=0;
unsigned int i;

    // make 16 bit words out of every two adjacent 8 bit
words in the packet
    // and add them up
   transmit_buffer[PACKET_OFFSET+1] = 0x00;
   transmit_buffer[PACKET_OFFSET] = 0x00 ;

    for (i=start;i<end;i=i+2){
        word16
=((transmit_buffer[i]<<8)&0xFF00)+(transmit_buffer[i+1]&0xF
F);
        sum = sum + (unsigned int) word16;
    }

    // take only 16 bits out of the 32 bit sum and add up
the carries
    while (sum>>16)
      sum = (sum & 0xFFFF)+(sum >> 16);

    // one's complement the result
    sum = ~sum;

   transmit_buffer[PACKET_OFFSET+1] = (unsigned char)(sum);
   transmit_buffer[PACKET_OFFSET] = (unsigned char)(sum
>>8) ;

    return sum;

}
```

```
void ethernet_interrupt_handler()
{
//printf("recv interrupt \n");
  Packet *pkt;
  unsigned int receive_status;
  int i;

  receive_status = ReceivePacket(receive_buffer,
&receive_buffer_length);

  /* Clear the DM9000A ISR: PRS, PTS, ROS, ROOS 4 bits, by
RW/C1 */
//  dm9000a_iow(ISR, 0x3F);

  /* Re-enable DM9000A interrupts */
//  dm9000a_iow(IMR, INTR_set);


  if (receive_status == DMFE_SUCCESS) {
//    printf("$$$$");
    for(i=0; i<receive_buffer_length;++i){
//        printf("%x ", receive_buffer[i] );
    }
//    printf("$$$$");

//    printf("\n\nReceive Packet Length = %d",
receive_buffer_length);
    for(i=0;i<receive_buffer_length;i++) {
    }

    if (receive_buffer_length >= 14) {
      //  A real Ethernet packet
      if (receive_buffer[12] == 8 && receive_buffer[13] ==
0 &&  receive_buffer_length >= 34) {
        // An IP packet
            if (receive_buffer[23] == 0x11) {
              // A UDP packet
printf("receive pckt ");

    if (receive_buffer_length >= UDP_PACKET_PAYLOAD_OFFSET
) {
        if( strncmp( (char*)receive_buffer +
UDP_PACKET_PAYLOAD_OFFSET, "CLUE", 4) !=0 )
          return ;

        pkt = (Packet*) malloc(sizeof(Packet));
```

```
        curMsgChar = UDP_PACKET_PAYLOAD_OFFSET +4;
        pkt->From = (unsigned
int)receive_buffer[curMsgChar];
        curMsgChar++;
        pkt->To = (unsigned int)receive_buffer[curMsgChar];
        curMsgChar++;
        pkt->packetType = (unsigned
int)receive_buffer[curMsgChar];
        curMsgChar++;

  switch(pkt->packetType) {

  case MasterBdcast:
        printf("master\n");

    pkt->u.s1.id = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;

    break;

  case JoinRequest:
        printf("join request\n");
    break;

  case PlayerAssigned :

    pkt->u.s2.assignedId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;

    printPacket(receive_buffer, receive_buffer_length);
    printf("player assigned %d %d\n", (unsigned
int)receive_buffer[curMsgChar-1], curMsgChar-1);


/*    pkt->u.s2.ip_address[0] = ;
    pkt->u.s2.ip_address[1] = receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s2.ip_address[2] = receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s2.ip_address[3] = receive_buffer[curMsgChar];
    curMsgChar++;
*/
    break;

  case GameInit:
```

```
        printf("game init\n");

    for( i=0; i<21 ; ++i){
      pkt->u.s3.playerId[i] = (unsigned
int)receive_buffer[curMsgChar];
      curMsgChar++;
      pkt->u.s3.cardIndex[i] = (unsigned
int)receive_buffer[curMsgChar];
      curMsgChar++;
    }
      pkt->u.s3.playerCount = (unsigned
int)receive_buffer[curMsgChar];
      curMsgChar++;

    break;

  case DiceRolled :
        printf("dice rolled\n");

    pkt->u.s4.value = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s4.playerId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;

    break;

  case Move:
        printf("move\n");

    pkt->u.s5.playerId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s5.oldPositionX = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s5.oldPositionY = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s5.newPositionX = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s5.newPositionY = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    break;
```

```
  case FinalPosition:
        printf("final position\n");

    pkt->u.s5.playerId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s5.oldPositionX = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s5.oldPositionY = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s5.newPositionX = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s5.newPositionY = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    break;

  case Suggestion:
        printf("suggestion\n");

    pkt->u.s6.playerId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s6.suspectId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s6.weaponId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s6.roomId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    break;

  case Disprove:
        printf("disprove\n");

    pkt->u.s7.senderPlayerId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s7.receiverPlayerId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
```

```
    pkt->u.s7.cardIndex = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    break;

  case ChangeTurn:
        printf("change turn\n");

    pkt->u.s8.senderPlayerId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s8.nextPlayerId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    break;

  case Accusation:
        printf("Accusation\n");

    pkt->u.s9.playerId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s9.suspectId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s9.weaponId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    pkt->u.s9.roomId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    break;

  case GameOver:
        printf("game over\n");

    pkt->u.s10.winnerId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
    break;

  case PlayerOut:
        printf("player out\n");

    pkt->u.s11.playerId = (unsigned
int)receive_buffer[curMsgChar];
    curMsgChar++;
```

```
    break;

    default: break;
  }


  updateDataOnPacketArrival(pkt);

        }
    } else {
          printf("Received non-UDP packet\n");
        }
      } else {
        printf("Received non-IP packet\n");
      }
    } else {
      printf("Malformed Ethernet packet\n");
    }

  } else {
    printf("Error receiving packet\n");
  }


  /* Display the number of interrupts on the LEDs */
  interrupt_number++;
//  outport(SEG7_DISPLAY_BASE, interrupt_number);

  /* Clear the DM9000A ISR: PRS, PTS, ROS, ROOS 4 bits, by
RW/C1 */
  dm9000a_iow(ISR, 0x3F);

  /* Re-enable DM9000A interrupts */
  dm9000a_iow(IMR, INTR_set);

printf("interrpt handler returns %d \n", pkt->packetType );
}


void sendPacket(Packet *pkt)
{
printf("sent Packet ");

    int i,packet_length;

    usleep(1000000);
  // reset data
```

```
  for (curMsgChar=MAX_MSG_LENGTH-1; curMsgChar>0;
curMsgChar--) {
    UDP_PACKET_PAYLOAD[curMsgChar] = 0;
  }

  strncpy( UDP_PACKET_PAYLOAD + curMsgChar, "CLUE", 4);
  curMsgChar+=4;

  UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>From;
//  printf("%d %c\n", pkt->From,
UDP_PACKET_PAYLOAD[curMsgChar] );

  curMsgChar++;
  UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt->To;
  curMsgChar++;

  switch(pkt->packetType) {

  case MasterBdcast:
        printf("master\n");

    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>packetType; //'0';
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s1.id;
    curMsgChar++;

    break;

  case JoinRequest:
        printf("join request\n");

    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>packetType; //'1';
    curMsgChar++;
    break;

  case PlayerAssigned :
        printf("player assigned\n");

    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>packetType; //'2';
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s2.assignedId; //'2';
```

```
printf("%d \n",  (unsigned
int)UDP_PACKET_PAYLOAD[curMsgChar] );
    curMsgChar++;
    printPacket( transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET
+ curMsgChar +1);
/*  UDP_PACKET_PAYLOAD[curMsgChar] = pkt-
>u.s2.ip_address[0];
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = pkt-
>u.s2.ip_address[1];
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = pkt-
>u.s2.ip_address[2];
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = pkt-
>u.s2.ip_address[3];
    curMsgChar++;
*/
    break;

  case GameInit:
       printf("game init\n");

    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>packetType; //'3';
    curMsgChar++;
    for( i=0; i<21 ; ++i){
      UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s3.playerId[i];
      curMsgChar++;
      UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s3.cardIndex[i];
      curMsgChar++;
    }
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s3.playerCount;
    curMsgChar++;

    break;

  case DiceRolled :
       printf("dice rolled\n");

    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>packetType; //'4';
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
```

```
>u.s4.value;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s4.playerId;
    curMsgChar++;

    break;

  case Move:
       printf("move\n");
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>packetType; //'5';
    curMsgChar++;

    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s5.playerId;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s5.oldPositionX;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s5.oldPositionY;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s5.newPositionX;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s5.newPositionY;
    curMsgChar++;
    break;

  case FinalPosition:
       printf("final position\n");
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>packetType; //'6';
    curMsgChar++;

    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s5.playerId;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s5.oldPositionX;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s5.oldPositionY;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
```

```
>u.s5.newPositionX;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s5.newPositionY;
    curMsgChar++;
    break;

  case Suggestion:
        printf("suggestion\n");
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>packetType; //'7';
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s6.playerId;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s6.suspectId;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s6.weaponId;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s6.roomId;
    curMsgChar++;
    break;

  case Disprove:
        printf("Disprove\n");
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>packetType; //'8';
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s7.senderPlayerId;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s7.receiverPlayerId;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s7.cardIndex;
    curMsgChar++;
    break;

  case ChangeTurn:
        printf("change turn\n");
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>packetType; //'9';
    curMsgChar++;
```

```
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s8.senderPlayerId;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s8.nextPlayerId;
    curMsgChar++;
    break;

  case Accusation:
       printf("accusation\n");
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>packetType; //'10';
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s9.playerId;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s9.suspectId;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s9.weaponId;
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s9.roomId;
    curMsgChar++;
    break;

  case GameOver:
       printf("Game over\n");
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>packetType; //'11';
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (char)pkt-
>u.s10.winnerId;
    curMsgChar++;
    break;

  case PlayerOut:
       printf("Player out\n");
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>packetType; //'12';
    curMsgChar++;
    UDP_PACKET_PAYLOAD[curMsgChar] = (unsigned char)pkt-
>u.s11.playerId;
    curMsgChar++;
    break;
```

```
    default : break;
  }

      transmit_buffer[IP_ID_OFFSET] =  IP_count >> 8;
      transmit_buffer[IP_ID_OFFSET+1] =  IP_count & 0xff;
      IP_count++;

      UDP_PACKET_PAYLOAD[curMsgChar] = 0; // Terminate the
string
      packet_length = UDP_PACKET_PAYLOAD_OFFSET +
curMsgChar; // 42 + msglenght
      if( packet_length < 64 ) {
        int i;
        for( i=packet_length ; i<63;++i) {
            UDP_PACKET_PAYLOAD[i]=0;
            ++packet_length;
            ++curMsgChar;

        }
      }
      transmit_buffer[16] = (packet_length-14 +1) >> 8;
      transmit_buffer[17] = (packet_length-14 +1) & 0xff;

      transmit_buffer[UDP_PACKET_LENGTH_OFFSET] =
(packet_length-34 +1) >> 8;
      transmit_buffer[UDP_PACKET_LENGTH_OFFSET + 1] =
(packet_length-34 +1) & 0xff;

      checksum(14,34,IP_CHECKSUM_OFFSET);

      if (TransmitPacket(transmit_buffer,
UDP_PACKET_PAYLOAD_OFFSET + curMsgChar + 1)==DMFE_SUCCESS)
{
//        printf("sent Packet\n");
//        printPacket(UDP_PACKET_PAYLOAD_OFFSET +
curMsgChar +1);
//            printf("\nMessage sent successfully: %s \n",
UDP_PACKET_PAYLOAD );
      } else {
            printf("\nMessage sending failed\n");
      }
      // reset data
      for (curMsgChar=MAX_MSG_LENGTH-1; curMsgChar>0;
curMsgChar--) {
            UDP_PACKET_PAYLOAD[curMsgChar] = 0;
      }
```

```
}


//void sendChangeTurnPacket() {}
```

b) VHDL Code

```
--Version : final
--Last updated: 5/6/2007
-- Tom and Khal


library IEEE;
use  IEEE.STD_LOGIC_1164.all;
use  IEEE.STD_LOGIC_ARITH.all;
use  IEEE.STD_LOGIC_UNSIGNED.all;
use  IEEE.NUMERIC_STD.all;
LIBRARY lpm;
USE lpm.lpm_components.all;

--Entity Declaration
ENTITY controller IS
      PORT(
            clk50     : IN    STD_LOGIC;

            VGA_CLK,
          VGA_HS,
          VGA_VS,
          VGA_BLANK,
          VGA_SYNC : OUT std_logic;
          VGA_R,
          VGA_G,
          VGA_B : OUT std_logic_vector(9 DOWNTO 0);

          avs_s1_clk        : in  std_logic;
          avs_s1_reset_n    : in  std_logic;
          avs_s1_read       : in  std_logic;
          avs_s1_write      : in  std_logic;
          avs_s1_chipselect : in  std_logic;
          avs_s1_address    : in  std_logic_vector(1 downto 0);
          avs_s1_readdata   : out std_logic_vector(31 downto 0);
          avs_s1_writedata  : in  std_logic_vector(31 downto 0)
            );
END controller;
ARCHITECTURE behavior OF controller IS
```

```
-- Signals used to Access and Read Video Memory
--  board ROM
      SIGNAL      board_address: STD_LOGIC_VECTOR(12 DOWNTO 0);
      SIGNAL      board_data: STD_LOGIC_VECTOR(8 DOWNTO 0);
--  font ROM
      SIGNAL      font_address_a: STD_LOGIC_VECTOR(14 DOWNTO 0);
      SIGNAL      font_address_b: STD_LOGIC_VECTOR(14 DOWNTO 0);
      SIGNAL      font_data_a: STD_LOGIC_VECTOR(9 DOWNTO 0);
      SIGNAL      font_data_b: STD_LOGIC_VECTOR(9 DOWNTO 0);
--  palette ROM
      SIGNAL      pallete_address: STD_LOGIC_VECTOR(9 DOWNTO 0);
      SIGNAL      pallete_data: STD_LOGIC_VECTOR(29 DOWNTO 0);

      SIGNAL row_address: std_logic_vector(12 DOWNTO 0):=
CONV_STD_LOGIC_VECTOR(0,13);
      SIGNAL col_address: std_logic_vector(12 DOWNTO 0):=
CONV_STD_LOGIC_VECTOR(0,13);


-- Video Display Signals
      SIGNAL red_data, green_data, blue_data :STD_LOGIC_VECTOR(9 DOWNTO
0);
      SIGNAL pixel_col_count, pixel_row_count: std_logic_vector(2
DOWNTO 0):= CONV_STD_LOGIC_VECTOR(0,3);
      SIGNAL Hcount : std_logic_vector(9 downto 0);
      SIGNAL Vcount : std_logic_vector(9 downto 0);
      SIGNAL EndOfLine, EndOfField : std_logic;
      SIGNAL vga_hblank, vga_hsync, vga_vblank, vga_vsync : std_logic;
      SIGNAL clk : std_logic := '0';


-- avalon signals
      SIGNAL reset: STD_LOGIC;

-- Game Registers
      TYPE reg_type IS ARRAY (63 DOWNTO 0) OF std_logic_vector (12
DOWNTO 0);
      SIGNAL reg : reg_type;
      TYPE sprite_type IS ARRAY (63 DOWNTO 0) OF std_logic_vector (7
DOWNTO 0);
      SIGNAL sprite : sprite_type;
      TYPE position_type IS ARRAY (7 DOWNTO 0) OF std_logic_vector (12
DOWNTO 0);
      SIGNAL position : position_type;
      SIGNAL position_a : position_type;
      SIGNAL position_b : position_type;
      TYPE data_type IS ARRAY (7 DOWNTO 0) OF std_logic_vector (8
DOWNTO 0);
      SIGNAL data : data_type;
      SIGNAL data_a : data_type;
      SIGNAL data_b : data_type;
      SIGNAL line1: std_logic_vector(12 DOWNTO 0);
      SIGNAL line2: std_logic_vector(12 DOWNTO 0);
      SIGNAL go: std_logic:= '1';
      SIGNAL go1: std_logic:= '1';
```

```
      SIGNAL sprite_count: std_logic_vector(4 DOWNTO 0):=
CONV_STD_LOGIC_VECTOR(0,5);
      SIGNAL comp: STD_LOGIC_VECTOR(3 DOWNTO 0);
      SIGNAL temp: STD_LOGIC_VECTOR(8 DOWNTO 0);
      SIGNAL      counter1: STD_LOGIC_VECTOR(6 DOWNTO 0);
      SIGNAL      counter2: STD_LOGIC_VECTOR(4 DOWNTO 0);

      constant HTOTAL       : integer := 800;
      constant HSYNC        : integer := 96;
      constant HBACK_PORCH  : integer := 48;
      constant HACTIVE      : integer := 640;
      constant HFRONT_PORCH : integer := 16;
    constant VTOTAL       : integer := 525;
      constant VSYNC        : integer := 2;
      constant VBACK_PORCH  : integer := 33;
      constant VACTIVE      : integer := 480;
      constant VFRONT_PORCH : integer := 10;


--Components Instantiation
COMPONENT board
      PORT
      (
            address           : IN STD_LOGIC_VECTOR (12 DOWNTO 0);
            clock       : IN STD_LOGIC ;
            q           : OUT STD_LOGIC_VECTOR (8 DOWNTO 0)
      );
end COMPONENT;

COMPONENT font
      PORT
      (
            address_a         : IN STD_LOGIC_VECTOR (14 DOWNTO 0);
            address_b         : IN STD_LOGIC_VECTOR (14 DOWNTO 0);
            clock       : IN STD_LOGIC ;
            q_a         : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
            q_b         : OUT STD_LOGIC_VECTOR (9 DOWNTO 0)
      );
end COMPONENT;

COMPONENT pallete
      PORT
      (
            address           : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
            clock       : IN STD_LOGIC ;
            q           : OUT STD_LOGIC_VECTOR (29 DOWNTO 0)
      );
end COMPONENT;




BEGIN

board_inst : board PORT MAP (
            address      => board_address,
            clock  => clk,
```

```
            q      => board_data
      );


font_inst : font PORT MAP (
            address_a    => font_address_a,
            address_b    => font_address_b,
            clock  => clk,
            q_a    => font_data_a,
            q_b    => font_data_b
      );

pallete_inst : pallete PORT MAP (
            address      => pallete_address,
            clock  => clk,
            q      => pallete_data
      );

  process (clk50)
  begin
    if clk50'event and clk50 = '1' then
      clk <= not clk;
    end if;
  end process;

 reset <= avs_s1_reset_n;

 process (avs_s1_clk)
 begin
   if avs_s1_clk'event and avs_s1_clk = '1' then
     if reset = '0' then

-- For suspects 1st->cards area,2nd->message area (accusation),3rd-
>board, 4th->message area(current player)
-- suspects
      -- suspect #0 => Plum
            sprite(0)<=CONV_STD_LOGIC_VECTOR(99,8);
            sprite(1)<=CONV_STD_LOGIC_VECTOR(99,8);
            sprite(2)<=CONV_STD_LOGIC_VECTOR(99,8);
            sprite(3)<=CONV_STD_LOGIC_VECTOR(99,8);
      -- suspect #1 => White
            sprite(4)<=CONV_STD_LOGIC_VECTOR(108,8);
            sprite(5)<=CONV_STD_LOGIC_VECTOR(108,8);
            sprite(6)<=CONV_STD_LOGIC_VECTOR(108,8);
            sprite(7)<=CONV_STD_LOGIC_VECTOR(108,8);
      -- suspect #2 => Green
            sprite(8)<=CONV_STD_LOGIC_VECTOR(117,8);
            sprite(9)<=CONV_STD_LOGIC_VECTOR(117,8);
            sprite(10)<=CONV_STD_LOGIC_VECTOR(117,8);
            sprite(11)<=CONV_STD_LOGIC_VECTOR(117,8);
      -- suspect #3  => Peacock
            sprite(12)<=CONV_STD_LOGIC_VECTOR(126,8);
            sprite(13)<=CONV_STD_LOGIC_VECTOR(126,8);
            sprite(14)<=CONV_STD_LOGIC_VECTOR(126,8);
            sprite(15)<=CONV_STD_LOGIC_VECTOR(126,8);
      -- suspect #4 => Scarlet
            sprite(16)<=CONV_STD_LOGIC_VECTOR(135,8);
```

```
            sprite(17)<=CONV_STD_LOGIC_VECTOR(135,8);
            sprite(18)<=CONV_STD_LOGIC_VECTOR(135,8);
            sprite(19)<=CONV_STD_LOGIC_VECTOR(135,8);
      -- suspect #5 => Mustard
            sprite(20)<=CONV_STD_LOGIC_VECTOR(144,8);
            sprite(21)<=CONV_STD_LOGIC_VECTOR(144,8);
            sprite(22)<=CONV_STD_LOGIC_VECTOR(144,8);
            sprite(23)<=CONV_STD_LOGIC_VECTOR(144,8);

--For weapons, 1st register->cards, 2nd->message,3rd->board
-- weapons
      -- weapon #0 => Pipe
            sprite(24)<=CONV_STD_LOGIC_VECTOR(153,8);
            sprite(25)<=CONV_STD_LOGIC_VECTOR(153,8);
            sprite(26)<=CONV_STD_LOGIC_VECTOR(153,8);
      -- weapon #1 => Wrench
            sprite(27)<=CONV_STD_LOGIC_VECTOR(162,8);
            sprite(28)<=CONV_STD_LOGIC_VECTOR(162,8);
            sprite(29)<=CONV_STD_LOGIC_VECTOR(162,8);
      -- weapon #2 => Rope
            sprite(30)<=CONV_STD_LOGIC_VECTOR(171,8);
            sprite(31)<=CONV_STD_LOGIC_VECTOR(171,8);
            sprite(32)<=CONV_STD_LOGIC_VECTOR(171,8);
      -- weapon #3 => REvolver
            sprite(33)<=CONV_STD_LOGIC_VECTOR(180,8);
            sprite(34)<=CONV_STD_LOGIC_VECTOR(180,8);
            sprite(35)<=CONV_STD_LOGIC_VECTOR(180,8);
      -- weapon #4 => Knife
            sprite(36)<=CONV_STD_LOGIC_VECTOR(189,8);
            sprite(37)<=CONV_STD_LOGIC_VECTOR(189,8);
            sprite(38)<=CONV_STD_LOGIC_VECTOR(189,8);
      --weapon #5 => CandleStick
            sprite(39)<=CONV_STD_LOGIC_VECTOR(198,8);
            sprite(40)<=CONV_STD_LOGIC_VECTOR(198,8);
            sprite(41)<=CONV_STD_LOGIC_VECTOR(198,8);

--For rooms, 1st register->cards, 2nd->message
-- rooms
      -- room #0 => Kitchen
            sprite(42)<=CONV_STD_LOGIC_VECTOR(9,8);
            sprite(43)<=CONV_STD_LOGIC_VECTOR(9,8);
      -- room #1 => BallRoom
            sprite(44)<=CONV_STD_LOGIC_VECTOR(18,8);
            sprite(45)<=CONV_STD_LOGIC_VECTOR(18,8);
      -- room #2 => Conservatory
            sprite(46)<=CONV_STD_LOGIC_VECTOR(27,8);
            sprite(47)<=CONV_STD_LOGIC_VECTOR(27,8);
      -- room #3 => Dining Room
            sprite(48)<=CONV_STD_LOGIC_VECTOR(36,8);
            sprite(49)<=CONV_STD_LOGIC_VECTOR(36,8);
      -- room #4 => Billard
            sprite(50)<=CONV_STD_LOGIC_VECTOR(54,8);
            sprite(51)<=CONV_STD_LOGIC_VECTOR(54,8);
      -- room #5 => Lounge
            sprite(52)<=CONV_STD_LOGIC_VECTOR(63,8);
            sprite(53)<=CONV_STD_LOGIC_VECTOR(63,8);
      -- room #6 => Hall
```

```
            sprite(54)<=CONV_STD_LOGIC_VECTOR(72,8);
            sprite(55)<=CONV_STD_LOGIC_VECTOR(72,8);
      -- room #7 => Library
            sprite(56)<=CONV_STD_LOGIC_VECTOR(81,8);
            sprite(57)<=CONV_STD_LOGIC_VECTOR(81,8);
      -- room #8 => Study
            sprite(58)<=CONV_STD_LOGIC_VECTOR(90,8);
            sprite(59)<=CONV_STD_LOGIC_VECTOR(90,8);

-- Accusation
            sprite(60)<=CONV_STD_LOGIC_VECTOR(234,8);

-- Suggestion
            sprite(61)<=CONV_STD_LOGIC_VECTOR(225,8);

--Extra Blank Registers
            sprite(62)<=CONV_STD_LOGIC_VECTOR(252,8);
            sprite(63)<=CONV_STD_LOGIC_VECTOR(255,8);

            reg(0) <= "1111111111111";
            reg(1) <= "1111111111111";
            reg(2) <= "1111111111111";
            reg(3) <= "1111111111111";
            reg(4) <= "1111111111111";
            reg(5) <= "1111111111111";
            reg(6) <= "1111111111111";
            reg(7) <= "1111111111111";
            reg(8) <= "1111111111111";
            reg(9) <= "1111111111111";
            reg(10) <= "1111111111111";
            reg(11) <= "1111111111111";
            reg(12) <= "1111111111111";
            reg(13) <= "1111111111111";
            reg(14) <= "1111111111111";
            reg(15) <= "1111111111111";
            reg(16) <= "1111111111111";
            reg(17) <= "1111111111111";
            reg(18) <= "1111111111111";
            reg(19) <= "1111111111111";
            reg(20) <= "1111111111111";
            reg(21) <= "1111111111111";
            reg(22) <= "1111111111111";
            reg(23) <= "1111111111111";
            reg(24) <= "1111111111111";
            reg(25) <= "1111111111111";
            reg(26) <= "1111111111111";
            reg(27) <= "1111111111111";
            reg(28) <= "1111111111111";
            reg(29) <= "1111111111111";
            reg(30) <= "1111111111111";
            reg(31) <= "1111111111111";
            reg(32) <= "1111111111111";
            reg(33) <= "1111111111111";
            reg(34) <= "1111111111111";
            reg(35) <= "1111111111111";
            reg(36) <= "1111111111111";
            reg(37) <= "1111111111111";
```

```
            reg(38) <= "1111111111111";
            reg(39) <= "1111111111111";
            reg(40) <= "1111111111111";
            reg(41) <= "1111111111111";
            reg(42) <= "1111111111111";
            reg(43) <= "1111111111111";
            reg(44) <= "1111111111111";
            reg(45) <= "1111111111111";
            reg(46) <= "1111111111111";
            reg(47) <= "1111111111111";
            reg(48) <= "1111111111111";
            reg(49) <= "1111111111111";
            reg(50) <= "1111111111111";
            reg(51) <= "1111111111111";
            reg(52) <= "1111111111111";
            reg(53) <= "1111111111111";
            reg(54) <= "1111111111111";
            reg(55) <= "1111111111111";
            reg(56) <= "1111111111111";
            reg(57) <= "1111111111111";
            reg(58) <= "1111111111111";
            reg(59) <= "1111111111111";
            reg(60) <= "1111111111111";
            reg(61) <= "1111111111111";
            reg(62) <= "1111111111111";
            reg(63) <= "1111111111111";
     else
       if avs_s1_chipselect = '1' then
           if avs_s1_write = '1' then
             reg(conv_integer(avs_s1_writedata(21 downto 16))) <=
avs_s1_writedata(12 downto 0);
          end if;
        end if;
      end if;
   end if;
 end process;


--fontgen: process (board_data, pixel_row_count, pixel_col_count)
--begin
    --if reset = '0' then
            --font_address_a <= (OTHERS => '1');
            --font_address_b <= (OTHERS => '1');

    --else
            font_address_a <=board_data & pixel_row_count &
pixel_col_count ;
            font_address_b <=temp & pixel_row_count & pixel_col_count ;
      --end if;
--end process fontgen;

pallete_address <= font_data_b when (go ='0' and font_data_b(3 downto
0) /= "1111") else font_data_a;
--L2:PROCESS(font_data_a, font_data_b, go)
  --BEGIN
     --IF go = '1' THEN
           --pallete_address <= font_data_a;
```

```
      --ELSE

            --IF font_data_b(0) = '1' and font_data_b(1) = '1' and
font_data_b(2) = '1' and font_data_b(3) = '1' THEN
                  --pallete_address <= font_data_a;
            --ELSE
                  --pallete_address <= font_data_b;
            --END IF;
      --END IF;
--END PROCESS L2;

L4:PROCESS (clk, reset)
  BEGIN
      IF reset = '0' THEN
            red_data <= (OTHERS => '0');
            green_data <= (OTHERS => '0');
            blue_data <= (OTHERS => '0');
      ELSIF clk'event AND clk = '1' THEN
            IF (vga_hblank = '1') or (vga_vblank = '1') THEN
                  red_data <= (OTHERS => '0');
                  green_data <= (OTHERS => '0');
                  blue_data <= (OTHERS => '0');
            ELSE
                red_data <= pallete_data(29 DOWNTO 20);
                  green_data <= pallete_data(19 DOWNTO 10);
                  blue_data <= pallete_data(9 DOWNTO 0);
            END IF;
      END IF;
END PROCESS L4;

--L5: process ( row_address, col_address)
  --begin
    --if reset = '0' then
      --board_address <= (others => '0');
    --else
            board_address <= row_address + col_address;
    --end if;
  --end process L5;

  HCounter : process (clk, reset)
  begin
    if reset = '0' then
      Hcount <= (others => '0');
    elsif clk'event and clk = '1' then
      if EndOfLine = '1' then
        Hcount <= (others => '0');
      else
        Hcount <= Hcount + 1;
      end if;
    end if;
end process HCounter;

EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';


VCounter: process (clk, reset)
  begin
```

```
    if reset = '0' then
      Vcount <= (others => '0');
    elsif clk'event and clk = '1' then
      if EndOfLine = '1' then
        if EndOfField = '1' then
          Vcount <= (others => '0');
        else
          Vcount <= Vcount + 1;
        end if;
      end if;
    end if;
end process VCounter;

EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';


-- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK
  HSyncGen : process (clk, reset)
  begin
    if reset = '0' then
      vga_hsync <= '1';
    elsif clk'event and clk = '1' then
      if EndOfLine = '1' then
        vga_hsync <= '1';
      elsif Hcount = HSYNC - 1 then
        vga_hsync <= '0';
      end if;
    end if;
  end process HSyncGen;

  HBlankGen : process (clk, reset)
  begin
    if reset = '0' then
      vga_hblank <= '1';
    elsif clk'event and clk = '1' then
      if Hcount = HSYNC + HBACK_PORCH then
        vga_hblank <= '0';
      elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
        vga_hblank <= '1';
      end if;
    end if;
  end process HBlankGen;

  VSyncGen : process (clk, reset)
  begin
    if reset = '0' then
      vga_vsync <= '1';
    elsif clk'event and clk = '1' then
      if EndOfLine ='1' then
        if EndOfField = '1' then
          vga_vsync <= '1';
        elsif Vcount = VSYNC - 1 then
          vga_vsync <= '0';
        end if;
      end if;
    end if;
  end process VSyncGen;
```

```
  VBlankGen : process (clk, reset)
  begin
    if reset = '0' then
      vga_vblank <= '1';
    elsif clk'event and clk = '1' then
      if EndOfLine = '1' then
        if Vcount = VSYNC + VBACK_PORCH - 1 then
          vga_vblank <= '0';
        elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
          vga_vblank <= '1';
        end if;
      end if;
    end if;
  end process VBlankGen;


  ColGen: process (clk, reset)
  begin
    if reset = '0' then
      pixel_col_count <= (others => '0');
      col_address <= (others => '0');
    elsif clk'event and clk = '1' then
            if (Hcount >= HSYNC + HBACK_PORCH) and (Hcount < HSYNC +
HBACK_PORCH + HACTIVE) then
                  if pixel_col_count < CONV_STD_LOGIC_VECTOR(7,3) then
                        pixel_col_count <= pixel_col_count + '1';
                  else
                        pixel_col_count <= (OTHERS => '0');
                        col_address <= col_address + '1';
                  end if;
            else
                  col_address <= (OTHERS => '0');
            end if;
      end if;
      end process ColGen;

      RowGen: process (clk, reset)
        begin
          if reset = '0' then
            pixel_row_count <= (others => '0');
              row_address <= (others => '0');
          elsif clk'event and clk = '1' then
                  if (Vcount >= VSYNC + VBACK_PORCH-1) and (Vcount <
VSYNC + VBACK_PORCH + VACTIVE-1) then
                        if pixel_col_count = CONV_STD_LOGIC_VECTOR(7,3)
and col_address = CONV_STD_LOGIC_VECTOR(79,10) then
                              if pixel_row_count <
CONV_STD_LOGIC_VECTOR(7,3) then
                                    pixel_row_count <= pixel_row_count
+ '1';
                              else
                                    row_address <= row_address +
CONV_STD_LOGIC_VECTOR(80,13);
                                    pixel_row_count<= (others => '0');
                              end if;
                        end if;
```

```
                else
                      row_address <= (OTHERS => '0');
                end if;

          end if;

          end process RowGen;



load_shifter : process (clk, reset)
      begin
          if reset = '0' then
                counter2 <= (others => '0');
                  position(0) <= "1100000000000";
                  position(1) <= "1100000000000";
                  position(2) <= "1100000000000";
                  position(3) <= "1100000000000";
                  position(4) <= "1100000000000";
                position(5) <= "1100000000000";
                  position(6) <= "1100000000000";
                  position(7) <= "1100000000000";
                  position_a(0) <= "1100000000100";
                  position_a(1) <= "1100000000000";
                  position_a(2) <= "1100000000000";
                  position_a(3) <= "1100000000000";
                  position_a(4) <= "1100000000000";
                position_a(5) <= "1100000000000";
                  position_a(6) <= "1100000000000";
                  position_a(7) <= "1100000000000";
                  position_b(0) <= "1100000000100";
                  position_b(1) <= "1100000000000";
                  position_b(2) <= "1100000000000";
                  position_b(3) <= "1100000000000";
                  position_b(4) <= "1100000000000";
                position_b(5) <= "1100000000000";
                  position_b(6) <= "1100000000000";
                  position_b(7) <= "1100000000000";

                  data(0) <= "111111111";
                  data(1) <= "111111111";
                  data(2) <= "111111111";
                  data(3) <= "111111111";
                  data(4) <= "111111111";
                data(5) <= "111111111";
                  data(6) <= "111111111";
                  data(7) <= "111111111";
                  data_a(0) <= "111111111";
                  data_a(1) <= "111111111";
                  data_a(2) <= "111111111";
                  data_a(3) <= "111111111";
                  data_a(4) <= "111111111";
                data_a(5) <= "111111111";
                  data_a(6) <= "111111111";
                  data_a(7) <= "111111111";
                  data_b(0) <= "111111111";
                  data_b(1) <= "111111111";
```

```
                    data_b(2) <= "111111111";
                    data_b(3) <= "111111111";
                    data_b(4) <= "111111111";
                  data_b(5) <= "111111111";
                    data_b(6) <= "111111111";
                    data_b(7) <= "111111111";

            elsif clk'event and clk = '1' then
                  if (vga_hblank = '1' and pixel_row_count = "000")
then
                        if counter2 <= 7 then
                              if reg(conv_integer(counter1)) >= line1
and reg(conv_integer(counter1)) < line2 then
                                    position(conv_integer(counter2))<=
reg(conv_integer(counter1));

      position_a(conv_integer(counter2))<=
reg(conv_integer(counter1))+"0000000000001";

      position_b(conv_integer(counter2))<=
reg(conv_integer(counter1))+"0000000000010";
                                    data(conv_integer(counter2))<=
sprite(conv_integer(counter1));
                                    data_a(conv_integer(counter2))<=
sprite(conv_integer(counter1))+"00000001";
                                    data_b(conv_integer(counter2))<=
sprite(conv_integer(counter1))+"00000010";
                                    counter2 <= counter2 + '1';
                              end if;
                              if (reg(conv_integer(counter1))+
CONV_STD_LOGIC_VECTOR(80,13) >= line1 and reg(conv_integer(counter1))+
CONV_STD_LOGIC_VECTOR(80,13) < line2) then
                                    position(conv_integer(counter2))<=
reg(conv_integer(counter1))+CONV_STD_LOGIC_VECTOR(80,13);

      position_a(conv_integer(counter2))<=
reg(conv_integer(counter1))+CONV_STD_LOGIC_VECTOR(81,13);

      position_b(conv_integer(counter2))<=
reg(conv_integer(counter1))+CONV_STD_LOGIC_VECTOR(82,13);
                                    data(conv_integer(counter2))<=
sprite(conv_integer(counter1))+"00000011";
                                    data_a(conv_integer(counter2))<=
sprite(conv_integer(counter1))+"00000100";
                                    data_b(conv_integer(counter2))<=
sprite(conv_integer(counter1))+"00000101";
                                    counter2 <= counter2 + '1';
                              end if;
                              if (reg(conv_integer(counter1))+
CONV_STD_LOGIC_VECTOR(160,13) >= line1 and reg(conv_integer(counter1))+
CONV_STD_LOGIC_VECTOR(160,13) < line2) then
                                    position(conv_integer(counter2))<=
reg(conv_integer(counter1))+CONV_STD_LOGIC_VECTOR(160,13);

      position_a(conv_integer(counter2))<=
reg(conv_integer(counter1))+CONV_STD_LOGIC_VECTOR(161,13);
```

```
      position_b(conv_integer(counter2))<=
reg(conv_integer(counter1))+CONV_STD_LOGIC_VECTOR(162,13);
                                    data(conv_integer(counter2))<=
sprite(conv_integer(counter1))+"00000110";
                                    data_a(conv_integer(counter2))<=
sprite(conv_integer(counter1))+"00000111";
                                    data_b(conv_integer(counter2))<=
sprite(conv_integer(counter1))+"00001000";

                                    counter2 <= counter2 + '1';
                                end if;
                            end if;
                        else
                          counter2 <= (others => '0');
                        end if;
                    end if;
end process load_shifter;
line1 <= row_address;
line2 <= row_address + CONV_STD_LOGIC_VECTOR(80,13);

counter_1 : process (clk, reset)
begin
      if reset = '0' then
            counter1 <= (others => '0');
      elsif clk'event and clk = '1' then
            if vga_hblank = '1' and pixel_row_count = "000" then
                  if counter1 < CONV_STD_LOGIC_VECTOR(63,7) then
                    counter1 <= counter1+'1';
                  end if;
          else
              counter1 <= (others => '0');
          end if;
      end if;
end process counter_1;


unloading_shifter : process (clk, reset)
begin
    if reset = '0' then
            temp <= "111111111";
            go <= '1';
      elsif clk'event and clk = '1' then
            if board_address = position(0) then
                temp <= data(0);
                  go <= '0';
            end if;

            if board_address = position_a(0) then
            temp <= data_a(0);
                  go <= '0';
            end if;

            if board_address = position_b(0) then
                temp <= data_b(0);
                  go <= '0';
            end if;
```

```
if board_address = position(1) then
    temp <= data(1);
      go <= '0';
end if;

if board_address = position_a(1) then
temp <= data_a(1);
      go <= '0';
end if;

if board_address = position_b(1) then
    temp <= data_b(1);
      go <= '0';
end if;

if board_address = position(2) then
temp <= data(2);
      go <= '0';
end if;

if board_address = position_a(2) then
    temp <= data_a(2);
      go <= '0';
end if;

if board_address = position_b(2) then
temp <= data_b(2);
      go <= '0';
end if;

if board_address = position(3) then
    temp <= data(3);
      go <= '0';
end if;

if board_address = position_a(3) then
    temp <= data_a(3);
      go <= '0';
end if;

if board_address = position_b(3) then
temp <= data_b(3);
      go <= '0';
end if;

if board_address = position(4) then
    temp <= data(4);
      go <= '0';
end if;

if board_address = position_a(4) then
temp <= data_a(4);
      go <= '0';
end if;

if board_address = position_b(4) then
```

```
            temp <= data_b(4);
              go <= '0';
          end if;

          if board_address = position(5) then
          temp <= data(5);
              go <= '0';
          end if;

          if board_address = position_a(5) then
            temp <= data_a(5);
              go <= '0';
          end if;

          if board_address = position_b(5) then
          temp <= data_b(5);
              go <= '0';
          end if;

          if board_address = position(6) then
            temp <= data(6);
              go <= '0';
          end if;

          if board_address = position_a(6) then
          temp <= data_a(6);
              go <= '0';
          end if;

          if board_address = position_b(6) then
            temp <= data_b(6);
              go <= '0';
          end if;

          if board_address = position(7) then
            temp <= data(7);
              go <= '0';
          end if;

          if board_address = position_a(7) then
          temp <= data_a(7);
              go <= '0';
          end if;

          if board_address = position_b(7) then
            temp <= data_b(7);
              go <= '0';
          end if;

          if  (board_address /= position(0) and board_address /=
position_a(0) and board_address /= position_b(0) and board_address /=
position(1) and board_address /= position_a(1) and board_address /=
position_b(1) and board_address /= position(2) and board_address /=
position_a(2) and board_address /= position_b(2) and board_address /=
position(3) and board_address /= position_a(3) and board_address /=
position_b(3) and board_address /= position(4) and board_address /=
position_a(4) and board_address /= position_b(4) and board_address /=
```

```
position(5) and board_address /= position_a(5) and board_address /=
position_b(5) and board_address /= position(6) and board_address /=
position_a(6) and board_address /= position_b(6) and board_address /=
position(7) and board_address /= position_a(7) and board_address /=
position_b(7)) then
                  temp <= "111111111";
                  go <= '1';
            end if;
      end if;
end process unloading_shifter;



VGA_CLK <= clk;
VGA_HS <= not vga_hsync;
VGA_VS <= not vga_vsync;
VGA_SYNC <= '0';
VGA_BLANK <= not (vga_hsync or vga_vsync);
VGA_R <= red_data;
VGA_G <= green_data;
VGA_B <= blue_data;


END behavior;
```