4/7/2004
Team pacman
Ke Xu, Eric Li, Winston Chao

Design of Pac-Man

Introduction: This game has been designed to include the components and techniques learned from class Labs dealing with the software side as well as with VHDL. Our version of Pac-man will be partially implemented with third party software; however, the graphics display of the game as well as most of the game state/controls will be entirely made by our team. The game portion of our project will be coded in C while the graphical portion of it will be done in VHDL.

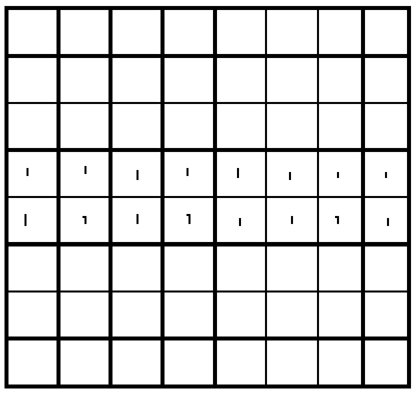
Game Description: Our version of Pac-Man will have several modifications to the original game. Enemy units will be configured to consume pellets along with the player unit. Our game is tentatively based upon a point system, if the player gobbles up over 50% of all pellets in the game, then the player wins. However, if the number of pellets eaten by all enemy units combined exceeds 50% of the total number, then the player automatically loses. Our game will host a maximum of two human players competing head to head. Additionally, our game state will contain "Magic Pellets" which will offer players special powers such as increased/decreased speed, invincibility, transparency…etc.

Software/Game State: The software side of our project will be coded in C. The game state will store the information of the game map along with the states of the player and enemy units. In-game movement will be controlled by a central-clock, movement will be scaled according to the cycles of the clock. For example, movement at normal speed would be one game tile per clock cycle, while increased movement will be 2 tiles/cycle, and decreased will be 1 tile/2 cycles. Our game will feature an auto-map generator which will basically generate the walls of a particular map along with pellet combinations and layout. The map-generator and the enemy units' AI code will be adapted from open-source C versions of Pac-Man found online. Our keyboard controls will be limited to 8 buttons (up, down, left, right,s,x,d,a). In every cycle, the particular key will be evaluated. This means that our player will have to ability to remain still in a game, it also means that continuous movement will require holding down a directional key. When multiple keys are pressed within the same cycle, our software will evaluate the most recent valid keystroke.

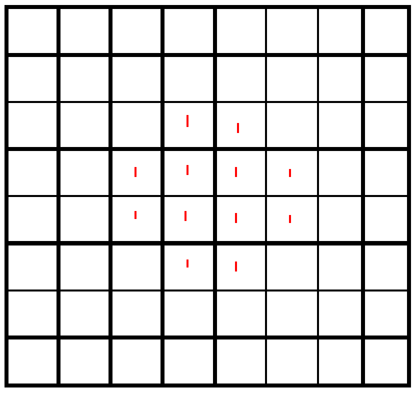
Hardware/VHDL: The basic graphical implementation of Pac-Man will borrow heavily from what we learned in Lab 5. Our game graphics will be done entirely within VHDL. A game state will be a matrix of tiles. Each kind of graphical tile will be stored in ROM. Each tile represents an 8x8 block of pixels on the display. The tile is implemented as a 2x9 array of Hexadecimal values, 2 rows of 1x8 elements of that array maps to the 8x8 pixel block, while the two elements at the beginning of each row codes for the color of the painted pixels. The tile is the basic building block of walls, pellets, and mobile units.

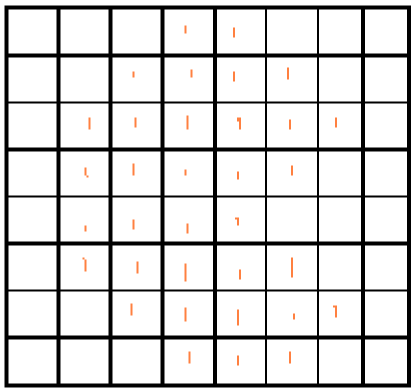
This particular implementation saves ROM space because of the color-coding header value. Additionally, our tiles are divided into 3 sets: wall, pellet, and unit. The permanent tiles are "wall" tiles that will never experience an in-game change. Wall tiles consist of two types: vertical and horizontal. Each wall tile is basically a tile with the two centering rows or columns painted. Temporary tiles are those that will change state in-game such as "pellet" and "unit" tiles. The graphics side will only paint the permanent tiles once at the beginning of the game; however, each temporary tile has to be repainted every cycle. Pellet tiles fall into 3 different states: uneaten, eaten, and occupied. We hope to differentiate between normal and magic pellet by color. Of course unit tiles are those whose physical locations on the display may change after each cycle. Thus the game state has to make the hardware repaint each unit after a time cycle. A unit tile breaks down into two types: Player and AI. Player tiles require 8 different states to correspond to the 4 directions and open/close of the mouth. AI tiles require only 2 states: open/close mouth. Different human player tiles will be differentiated by color. When a unit tile's location is the same as a pellet tile's location, the pellet's state is set to eaten, and afterwards the pellet tile will no longer show the actual pellet. The total number of pellet tiles along with initial pellet type cannot be changed, so a new pellet cannot be repainted in the tile of an eaten pellet.

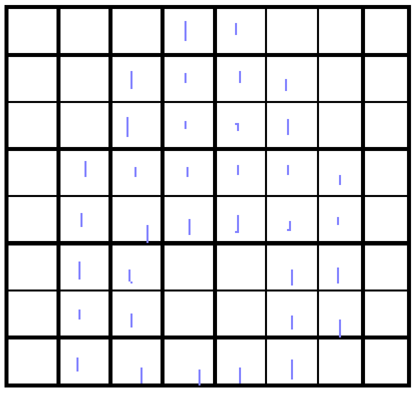
Illustrations:



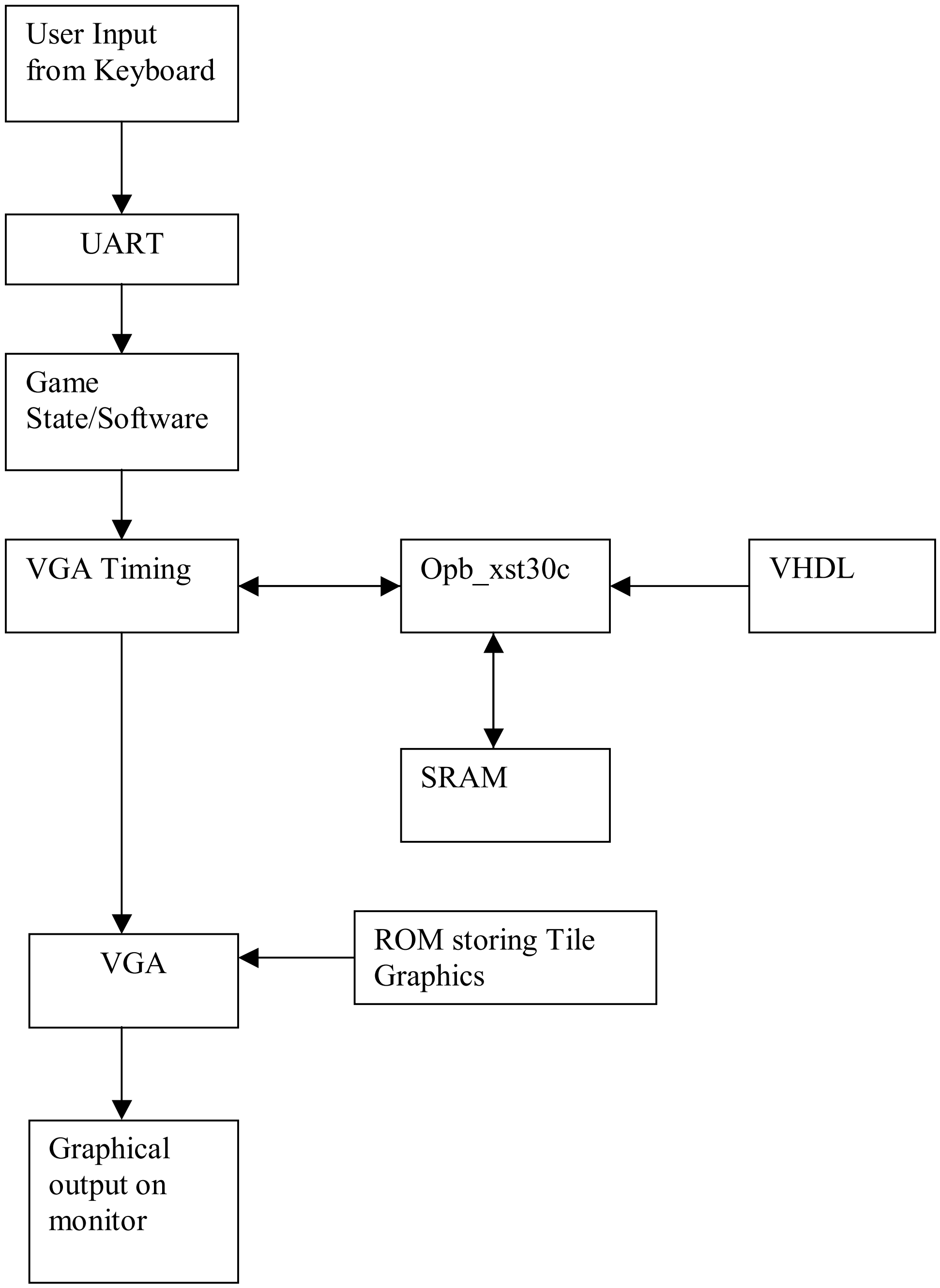
Horizontal Wall


Pellet Tile


Player Tile


Enemy Unit

Block Diagram:


User Input from Keyboard
UART
Game State/Software
VGA Timing
Opb_xst30c
VHDL
SRAM
VGA
ROM storing Tile Graphics
Graphical output on monitor