UF UNIVERSITY of FLORIDA
Herbert Wertheim
College of Engineering

Computer Engineering
CEN3907C

# Buffered Memory

## Overview

Many mobile and embedded systems operate on memory with limited performance and/or features. Even when hardware is very capable, optimization of memory operations can result in significant improvement in application performance. In this activity, students will work in an emulated x86 real mode environment (DOS) to build and run a simple “Find Four” (generic *Connect Four*) game using optimized memory reading and writing strategies. This application will make use of the mouse and VGA drivers from the “Programmed IO” lab assignment. The application will read from the mouse via polling and write directly to video memory to display the game state graphically on-screen.

### Structure

This project involves three major components:

1) Loading data files to prepare images in memory and the color palette in the graphics adapter
2) Implementing buffering and state-tracking strategies to track input and display output
3) Building the program / game logic to implement the “Find Four” game

While no specific source code breakdown is required, students are required to provide a Makefile to build their sources using the Open Watcom C/C++ toolchain suite (as provided).

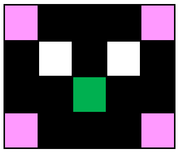
### Data File Formats

This activity involves reading two types of data, a palette file and a VGA image file.

The palette file (**findfour.pal**) is always exactly 768 bytes; it contains a sequence of 3-byte pairs representing the RGB values (one byte each) for color 0, color 1, … color n. The palette RGB values are always 6-bit values (0-63), with the highest two bits always off for simplicity.

| Color 0 | | | Color 1 | | | Color 2 | | | Color 3 | | | Color 4 | | | Color 5 | | | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | G | B | R | G | B | R | G | B | R | G | B | R | G | B | R | G | B | … |

The image files hold data on the sprites – the board block (**block.vga**), blue chip (**blue.vga**), and red chip (**red.vga**). The first byte of data is the width; the second the height; and the remaining are the color numbers (1 byte per pixel), row-by-row. Here’s an example from a 5x4 image:

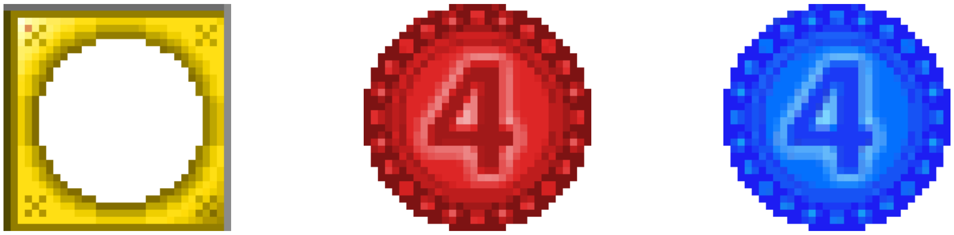


***Image***

| W | H | Row 0 | | | | | Row 1 | | | | | Row 2 | | | | | Row 3 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 5 | 00 | 01 | 01 | 01 | 00 | 03 | F0 | 01 | F0 | 01 | 01 | 01 | 22 | 01 | 01 | 00 | 01 | 01 | 01 | 00 |

5 bytes

When loaded correctly, the images will appear as shown. Color 0 is treated as transparent.

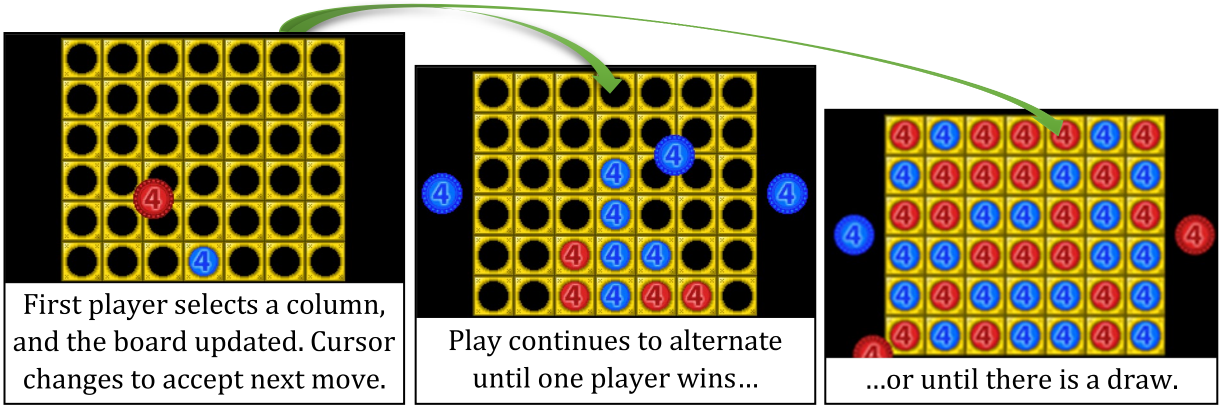


# Specification

The game should proceed according to the gameplay rules. The implementation will include double-buffering and the dirty rectangles strategies for memory handling. Structurally, the program's function is broken into initialization, update loop, and gameplay logic.

## Gameplay

Players take turns adding chips to the board, which drop to the lowest open row. The board is made up of a 7-column x 6-row grid. Once one player achieves four vertical, horizontal, or diagonal chips in a row, that player wins. Play proceeds as follows:



First player selects a column, and the board updated. Cursor changes to accept next move.

Play continues to alternate until one player wins…

…or until there is a draw.

## Drawing / Updating the Board

The Find Four game board is composed of the chips and the block structure the chips drop into. The size of the blocks and chips images are exactly 32x32. The board's upper-left-hand corner begins drawing at (x, y) position (48, 4). First, the chips (if any) should be drawn, and then the block image should be laid over it (respecting transparent pixels). Each chip-and-block is spaced 32 pixels apart on both the x-axis and y-axis.

Students will use double-buffering for video memory updates. This will require a back buffer of video memory – 64,000 bytes (one per pixel) – on which all image operations will occur. This will be copied to the front buffer (video memory) once per update cycle. Additionally, the **dirty rectangles** approach should be used to update the back buffer each frame – updating only tiles / chips that have changed (been added), rather than rebuilding the board on each update. This saves time and memory operations compared to re-drawing the entire board each time, speeding up the update process.

# Provided API

The following classes are provided in the **drivers.lib** file, with declarations in **Mouse.h** and **Vga.h**.

## Mouse Class

The **Mouse** class represents the mouse device and provides methods to interact with it.

```
public static Mouse *getInstance()
```

Returns an instance to the **Mouse** object. If necessary, this method initializes the driver. If no mouse is present, this method returns a null pointer (**NULL**, or **0**).

```
public static void shutdown()
```

Shutdown and clean up the **Mouse** object, if allocated.

```
public uint16_t getX()
public uint16_t getY()
```

Returns the *Mouse*’s X / Y coordinate.

```
public bool getLeft()
public bool getMiddle()
public bool getRight()
```

Returns the state of the Left / Middle / Right button of the *Mouse*.

```
public void reset(int xLimit, int yLimit, int mickeyScale)
```

Sets the *Mouse*‘s maximum X and Y values and the number of mickeys per coordinate increment.

```
public void update()
```

Captures latest hardware movement and button states, integrating them into the *Mouse*‘s state.

## Vga Class

This static class represents the VGA connector. **All methods are public and static.**

```
uint8_t *getAddress()
```

Returns a pointer the video memory (large memory model).

```
void getPaletteEntry(uint8_t number, uint8_t *r, uint8_t *g, uint8_t *b)
```

Fetches the color **number** and stores its red, green, and blue values at addresses of **r**, **g**, and **b**.

```
void setPaletteEntry(uint8_t number, uint8_t r, uint8_t g, uint8_t b)
```

Stores the **r**, **g**, and **b** values in the palette as the color indicated by **number**.

```
void setPaletteMask(uint8_t mask)
```

Sets the VGA adapter’s palette **mask** value.

```
void setMode(uint8_t mode)
```

Sets the VGA adapter’s graphical **mode**.

```
void verticalSync()
```

Delays until a vertical synchronization has occurred, then returns.

## Initialization

When the program begins, these steps should be completed to initialize the program.

1) Verify that the mouse is available, and if not, terminate
2) Load the palette and image files, terminating on error
3) Change to graphical mode (0x13)
4) Load the palette into the VGA controller
5) Clear the screen and back buffer using Color 0, then draw the game board on the back buffer
6) Initialize the mouse position to the center of the screen
7) Set the mouse cursor to be the chip of the ***blue*** player (the starting player)
8) Draw the cursor on the back buffer, centered on the mouse position
9) Set the current player to ***blue*** and start the update loop

## Update Loop

While the program is active, after initialization, the following steps should be done continuously.

1) Erase the area where the mouse cursor is currently drawn on the back buffer
2) Redraw the board only in the area potentially impacted (hint: it's a 2x2 block area)
3) If the game is over, display winning player's chip at locations (8, 84) and (280, 84). (For a tie, draw the blue chip on the left and the red chip on the right.)
4) Fetch updated mouse data from the mouse driver (buttons and position)
5) If the right mouse button is pressed, terminate the program immediately.
6) Draw the mouse cursor at the new location on the back buffer (centered)
7) Copy the back buffer to video memory
8) If the game is still in play, execute the gameplay logic

## Gameplay Logic

The gameplay logic is run on each iteration after initialization while the game has not ended.

1) If left button was not clicked in the previous iteration, but is now, remember the column.
2) If left button was pressed in previous iteration, but is now released:
    a. If the column is different from the remembered column, ignore and continue.
    b. If the column is the same, attempt to insert a chip in the current column.
        i. If the chip is successfully inserted…
            1. Add the chip and check for end of game.
            2. If the game is not over, change the current player and cursor.
        ii. Otherwise, make no change – play continues for current player.

# Submission

Your submission will be composed of the following elements:

- Screencast of your game running, demoing all features, as an unlisted Internet resource
- Report (**ProgrammedIOReport.pdf**) on Canvas, *including the link to your screencast*
- Source and build files as a zip archive file (**findfour.zip**) on Canvas

## Screencast

Your video demonstration will be submitted on Canvas as a list in your report, must be no more than 2 minutes, and should include the following, at a minimum:

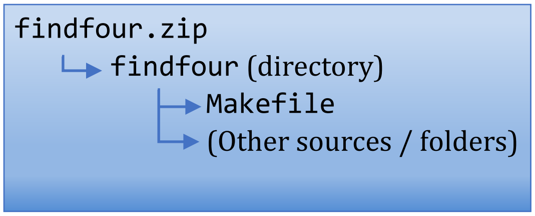
- Display the file directory/directories for the project, including the Makefiles
- Show and explain the contents of all Makefiles in your build
- Build the project to completion to show the build works
- Run the program generated by the build process and demonstrate all features

## Report

Your report will explain the process you used to develop the program, including what tools you used. It will describe how testing was performed and any known bugs. The report should be 500 words or fewer, cover all relevant aspects of the project, and be organized and formatted professionally – ***this is not a memo!***

## Compressed Archive (findfour.zip)

Your compressed file should have the following directory/file structure (including the library):

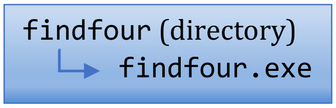


```
findfour.zip
  └→ findfour (directory)
       ├→ Makefile
       └→ (Other sources / folders)
```

To build the library, we will execute these commands (from the Dev Prompt):

```
V:\project$ unzip findfour.zip
V:\project\findfour$ cd findfour
V:\project\findfour$ wmake
```

The build process should generate one executable in the **`findfour`** directory:



```
findfour (directory)
  └→ findfour.exe
```