CS 241
Data Organization using C
Project 3: Adversarial Fill

Instructor: **Joel Castellanos**
e-mail: joel@unm.edu
Web: http://cs.unm.edu/~joel/
Office:
Farris Engineering Center
Room 319

5/5/2015

Quiz: Bitwise OR Operator

```
1. #include <stdio.h>
2.
3. void main(void)
4. {
5.   printf("%d\n", 39 | 15);
6. }
```

The output is:
a) **26**
b) **39**
c) **41**
d) **42**
e) **47**

	128	64	32	16	8	4	2	1
	0	0	1	0	0	1	1	1
|	0	0	0	0	1	1	1	1
	0	0	1	0	1	1	1	1

2

What is the output?

```
#include <stdio.h>
void main(void)
{
  char word[] = "gravity";
  *word = (*word -'a') + 'A';
  printf("%s\n", word);
}
```

a) **Gravity**
b) **Aravity**
c) **A**
d) **grAvity**
e) Might be segmentation fault, might run but with unpredictable output.

3

Normal Flood Fill Algorithms

- ***Flood fill*** is an algorithm that determines the area connected to a given node in a multi-dimensional array.
- Flood fill is used in the "bucket" fill tool of paint programs to fill connected, similarly-colored areas.
- Flood fill is also used in games such as Go and Minesweeper for determining which pieces are cleared.
- http://en.wikipedia.org/wiki/Flood_fill

4

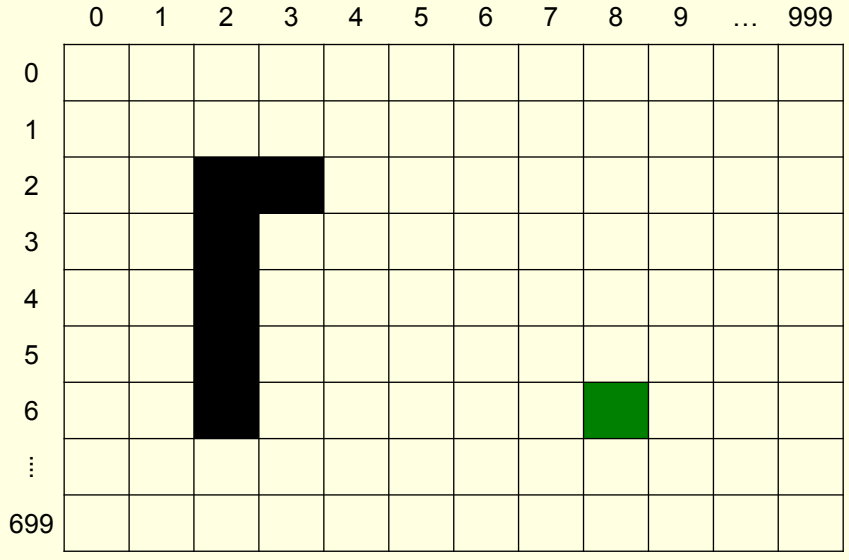
Four-way flood fill using a ***queue*** verses a ***stack*** for storage.

Adversarial Fill: Initial State

1) The board is a rectangular grid of 500 × 300 cells.
2) In each *game*, there are 1 through 6 players.
3) The initial state is:
   a) A four-direction, connected area of white cells.
   b) Random islands of black cells covering less than 10% of the total grid area.
   c) One cell per player in a unique color.
   d) No initial player cell will be within a Manhattan distance of ten from any boundary, island or other player cell.

5

Board Cell Numbering



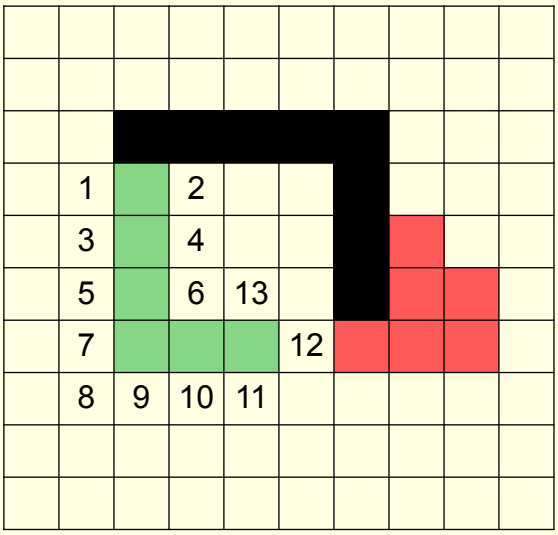
6

Legal Moves

On the next "*turn*", green may color any uncolored cell that is 4-direction adjacent to any currently green cell.

In the figure to the right, there are 13 such cells.

Cell #12 could be colored by green ***or*** red (not both).



7

MCP / Player Interface

1) Each player has 3 "public" methods: `_init()`, `_move()`, `_getName()`

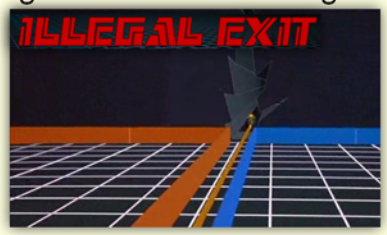
2) At the start of a new game, the MasterControlProgram (MCP) calls each team's `_init()` function giving it the game configuration:
   a) Each player's name and color.
   b) Each player's initial cell coordinates.
   c) The coordinates of all black wall cells.

3) Each timestep, the MCP will check to see which teams have returned from the last call to `_move()`. Any returned moves are validated and the game state updated. Then, in random order, all players who returned from the last call to `_move()` have their `_move()` called again.
   - ***Telling*** each player all changes since the last call to `_move()`.
   - ***Asking*** each player where he or she wants to move.

8

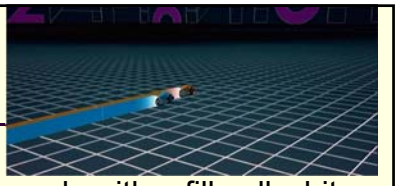
AFF *Spring 2015:* Game Rules

1) Only the MCP may call a player's `_init()`, `_move()`, or `_getName()` functions.
2) Any player that causes the program to exit with a runtime error is disqualified.
3) Any invalid moves are ignored.
4) Player code that can trick the MCP without causing the program to crash might be able to give itself an advantage.



9

AFF Grading Overview



[+20 Points]: In single player mode, your algorithm fills all white spaces in no more than:

whiteCellCount×TIMESTEP_MS + **`FIRST_TURN_MS`** seconds

[+20 Points]: In two player mode, your algorithm beats a standard breath-first fill 4 out of 5 times.

[+30 Points]: In six player mode, your algorithm beats five standard breath-first fill algorithm 4 out of 5 times.

[+30 Points]: In two player mode, your algorithm beats, 4 out of 5, the rather dumb ATTACK bots.

10

AFF *Spring 2015:* Extra Credit

Tourney
Thursday
May 7
10:00 - noon
FEC 309
Snacks will be served

+10 Points: Tournament Finalist.

+25 Points: Bronze tournament Winner.

+50 Points: Silver tournament Winner.

+100 Points: Gold tournament Winner.

11

static: Keep Your Private Parts Private

- The C programming language predates Object Oriented Programming and only supports four levels of visibility:
  - Program Scope,
  - File Scope,
  - Function Scope,
  - Block Scope

Use **`static`** to protect your variables and function calls from other teams.

- To avoid name clashes with other players, you must:
  - Declare all your non-public functions as static.
  - Declare all your persistent variables as static.
- In C, a function scope variable declared as **`static`** is initialized once, and retains its value between function calls.
- A ***function*** or ***variable*** declared **`static`** is visible ***only in that file***.

12

`tron.h`: Program Scope Constants

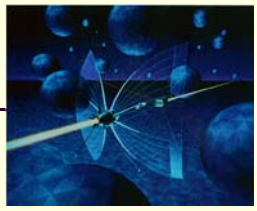
```
#define GRID_WIDTH  500
#define GRID_HEIGHT 300

#define MAX_PLAYERS 6
#define NORTH 1
#define EAST 2
#define SOUTH 4
#define WEST 8
#define MAX_MSG_LEN 10          //not including \0
#define MAX_PLAYER_NAME_LEN 15 //not including \0
#define FIRST_TURN_MS 2000
#define TIMESTEP_MS 5

              //   0        1       2     3    ...
enum ColorEnum {MAUVE,  GAMBOGE, MOSS,  ECRU,
                COBALT, IRIS,    BLACK, WHITE};
```

13

Interface between MCP and AI



Each student must implement the three, non-static (*"public"*) functions:

```
char* firstname_lastname_getName()

void firstname_lastname_init(struct InitData *data)

void firstname_lastname_move(struct MoveData *data)
```

Important: Do not save the addresses:

```
struct InitData *data
struct MoveData *data
```

The memory will be freed after your function returns.
Any *values* you want to save must be copied ***value-by-value*** to `static` local or `static` file-scope variables or structures.

14

tron.h: _getName

char* *firstname_lastname*_**getName()**

Returns a pointer to a character array containing your player name.

The returned name may contain any non-whitespace, printable ASCII character (base-10 codes 33 through 126).

If more than one student in the class returns a particular name, the MCP will force each to be unique by appending numbers.

A player's **_getName()** function must always return the same name.

When **_getName()** returns, the MCP will copy the name into its own storage and never again use the returned pointer.

No more than **MAX_PLAYER_NAME_LEN** characters (not including the terminating **'\0'**) will be copied from the returned pointer.

Name must be NULL-terminate.

15

tron.h: struct Cell

```
// The Cell structure is passed as part of both InitData,
//    and MoveData. Also, Cell is returned by MoveData.

struct Cell
{
  enum ColorEnum color;
  short x;                  //[0, GRID_WIDTH)
  short y;                  //[0, GRID_HEIGHT)
  struct Cell* next;        // ==0 if last in list.
};
```

16

tron.h: struct Player

```
// The Player structure is only
//      used in InitData.
// Since at the game start, each player is given exactly one
//      cell, the startCell linked list will contain only one
//      element.

struct Player
{
  char* name;              //null terminated.
  struct Cell* startCell;
};
```

17

tron.h: struct InitData

```
struct InitData
{ int playerCount; //[1, 6]
  int gridWidth, gridHeight;

  //Array of pointers to each player
  struct Player** playerList;


  //wallList is a 1D linked list of
  //     wall cells.
  //The list is sorted: island, x, y.
  struct Cell* wallList;
};
```

playerList[0] ->
playerList[1] ->
playerList[2] ->
⋮
playerList[n] ->

18

tron.h: struct MoveData

```
struct MoveData
{
  int turnNumber; // == 1 on first move of game.
  char* msg;    // Player may optionally return flavor text.

  //changeList is a linked list of all cells that have been
  //    colored since the last call to _move() returned from
  //    the player to whom this structure is being passed.
  struct Cell* changeList;

  //AI must copy the x and y values of the cell which it wants
  //   to color. The color and next fields are ignored.
  struct Cell* myMove;
};
```

19

Compiling AFF

1) Place your AI code in the file:
   yourFirstName_yourLastName`.c`
2) In a clean directory with your source code, copy from the class website the files **`tron.h`**, **`libmcp.a`** (the MCP library), and **`classDummy.c`**.
3) Edit **`classDummy.c`** by commenting out each **`_getName()`**, **`_init()`** and **`_move()`** method with you name.
4) Compile all **`.c`** files in the directory and link with the static MCP library, **`libmcp.a`**:

   ```
   gcc *.c -L. -lmcp -lm -lpthread -o AFFGame
   ```

   This compiles all **`.c`** files in the directory, links with **`libmcp.a`** and creates a stand-alone executable with the name **`AFFGame`**.

20

Running the MCP with your AI

- After you have compiled your code and linked with the MCP, run by entering the command:

  `./AFFGame`

- With no arguments, **`AFFGame`** will display a usage screen telling you what arguments to use to specify the, an optional random number seed, which players are to compete, etc.
- When **`AFFGame`** runs successfully with correct arguments, it will produce a file that contains all moves of the game: **`movelist.aff`**
- (coming soon) running **`AFFGame`** will also display a realtime SDL (Simple DirectMedia Layer) window.

21

Representing Direction

```
#define NORTH 1
#define EAST 2
#define SOUTH 4
#define WEST 8
```

given in **`tron.h`**

By using powers of 2, one integer variable can represent more than one direction. For example:

```
static int getDirBits(int x, int y)
{
  int dirBits = 15;  //All 4 flags on: =N|E|S|W
  if (grid[x][y-1] != WHITE) dirBits -= NORTH;
  if (grid[x][y+1] != WHITE) dirBits -= SOUTH;
  if (grid[x-1][y] != WHITE) dirBits -= EAST;
  if (grid[x+1][y] != WHITE) dirBits -= WEST;
  return dirBits;
}
```

Why not use if, else if, else?

22

Use Many Small Helper Functions

It is often useful to know the number of choices into which a particular cell may.

```
static int getNumberOfOpenDirections(int
   dirBits)
{
  int openCount = 0;
  if (dirBits & UP) openCount++;
  if (dirBits & DOWN) openCount++;
  if (dirBits & LEFT) openCount++;
  if (dirBits & RIGHT) openCount++;
  return openCount;
}
```

23

Unit Test your Helper Functions

```
static void test_getNumberOfOpenDirections()
{
  // 1) Clear the grid.
  // 2) Set, using a short series of assignment statements,
  //       a **few** specific walls and player colored cells
  //       showing in a small area  of the grid.
  // 3) Call getNumberOfOpenDirections() on the
  //       various cases and print results.
  // 4) Verify your results with hand drawing on graph paper.
}
```

24

Test Small!

- When developing your AI, change the grid size to something you can print out (maybe 20x20) and actually read.
- After you have the bugs worked out on a small size, THEN try the full size.
- When you find a bug on a full size grid, go back to a small size and try to reproduce it.

25

Gotta Love `printf`

If you have an error, try seeing if you can recreate it on a small size grid.
Try printing all values in your grid just before and just after a crash.

```
static void printGrid(void)
{ int x, y;
  for (y=0; y<gridSize; y++)
  { for (x=0; x<gridSize; x++)
    {
      printf("%2d ", grid[x][y]);
    }
    printf("\n");
  }
}
```

26

Gotta Love `printf`

```
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 6 6 6 6 6 6 6 7 7 7 7 7 7
7 7 6 6 6 6 6 6 6 0 7 7 7 7 7
7 7 7 7 7 6 6 7 0 0 0 7 7 7 7
7 7 7 7 7 6 6 7 7 0 0 0 3 7 7
7 7 7 7 7 6 6 7 7 0 0 7 3 3 3
7 7 7 7 7 7 7 7 7 0 7 3 3 3 3
7 7 7 7 7 7 7 7 7 7 7 7 3 7 3
7 7 7 7 7 7 7 7 7 7 7 7 3 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
```

27