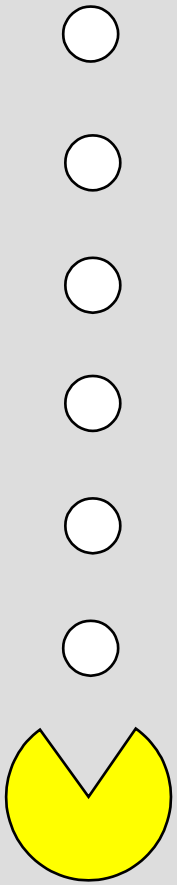
Basic Video Game Development



Sonoma State University
Computer Science Colloquium
September 11, 2003

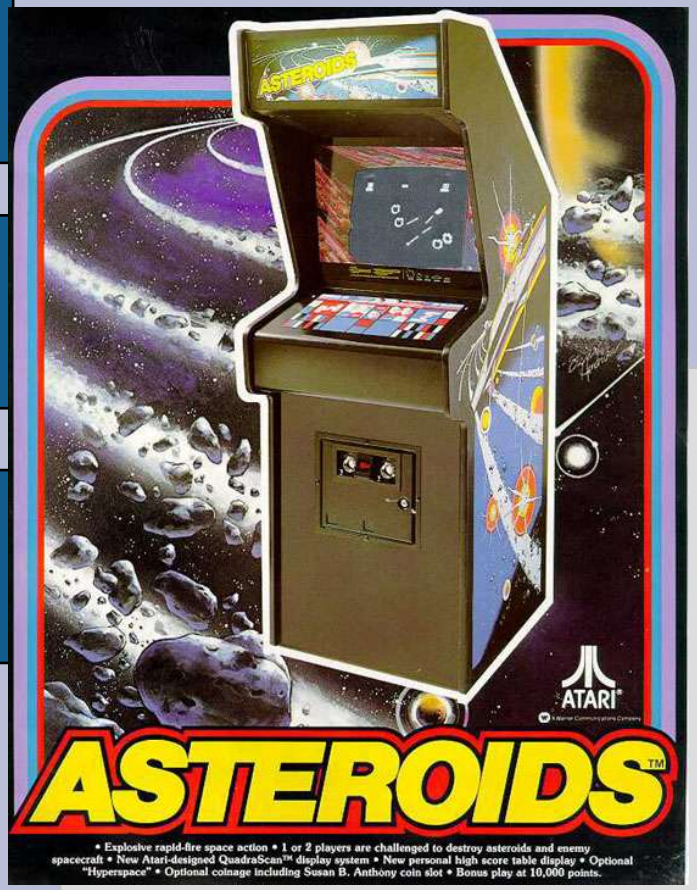
Presented by:
Bill Kendrick
New Breed Software
Davis, California

Where did I start?

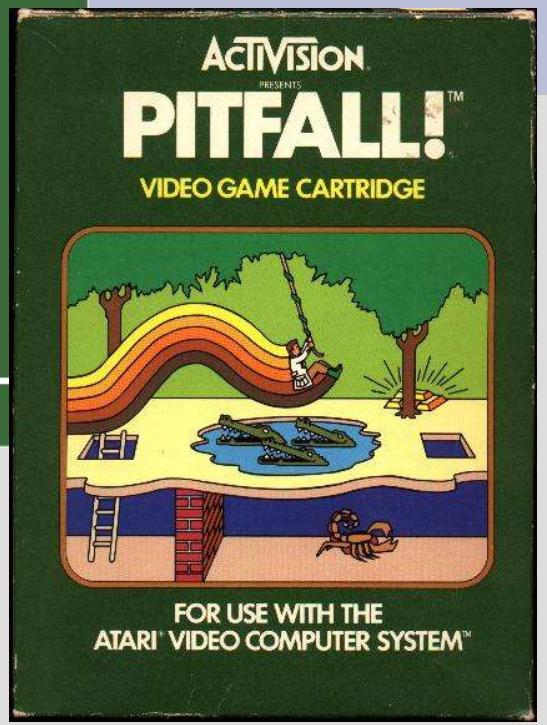
- Atari 1200XL
  8-bit computer
  (48KB RAM, 5¼” 128KB disks)
- BASIC programming language
- Self-taught from books and magazines

Console/arcade games of the era

- Shooting games
  Asteroids, Battlezone, Riverraid, Defender
- Racing games
  Pole Position
- Sports games
  Soccer, football, baseball, hockey
- Puzzle games
  Chess, Checkers, Reversi
- Jump-n-run games
  Pitfall, Lode Runner, Jumpman
- Hard to define
  Pac-Man, Human Cannonball, Frogs 'n Flies


ASTEROIDS
ATARI
ASTEROIDS™
• Explosive rapid-fire space action • 1 or 2 players are challenged to destroy asteroids and enemy spacecraft • New Atari-designed QuadraScan™ display system • New personal high score table display • Optional "Hyperspace" • Optional coinage including Susan B. Anthony coin slot • Bonus play at 10,000 points.


0
45:00
0


ACTIVISION
PRESENTS
PITFALL!™
VIDEO GAME CARTRIDGE
FOR USE WITH THE
ATARI® VIDEO COMPUTER SYSTEM™


150


04
00

What do they have in common?

- Easy to learn
- Easy to play
- Runs on slow hardware with limited RAM

Why do we care today?

- Easy to learn
  Larger potential audience
- Easy to play
  More 'repeat customers'
- Runs on slow hardware with limited RAM
  Handheld devices, cellphones, web browsers!

Plus, they're FUN!

Let's get started

- **Step 1:**
  What is the game about?
- **Step 2:**
  What environment(s) is it expected to run in?
- **Step 3:**
  ???
- **Step 4:**
  Profit!

“Stupid Joke” Game

- What is the game about?

  Q: “Why did the chicken cross the road?”
  A: “To get to the other side!”

  You play the chicken. Your objective is to cross a busy freeway to earn points.

Note: There *is* such a game. Activision's “Freeway” for the Atari 2600 game console, designed and written by David Crane, of “Pitfall” fame!

Where will it run?

(The game, not the chicken!)

- Modern computers
  (because they're easy to develop for)
- Keyboard or joystick control
  (not mouse)
- Possibly handheld systems
  (so be concerned about inputs!)

Let's get coding!

- C programming language
  Well supported
  Free compilers for various platforms & OSes
  Simple to program
  (It's all Bill knows!)

- Simple DirectMedia Layer
  GNU Library General Public License (LGPL)
  Runs on various platforms & OSes
  Simple to program
  Written in C! (has other 'bindings,' too)

What will we have?

- Chicken
  Controlled by player's keyboard/joystick

- Cars
  Automatically controlled;
  Various densities, speeds, and speed changes, based on difficulty level

Seriously... that's about all there is!

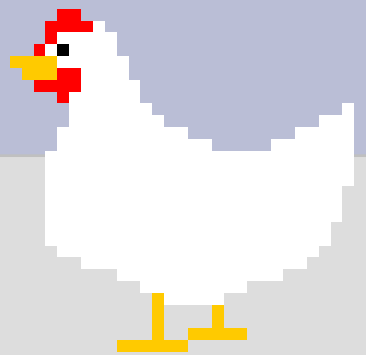
Boring program initialization

```
                                                              ;

if (SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO) < 0)
{
   fprintf(stderr, "Error init'ing SDL:  %s\n",
           SDL_GetError());
   exit(1);
}
```

Open Display

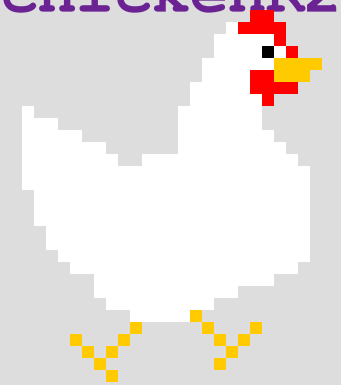
```
SDL_Surface * screen;

screen = SDL_SetVideoMode(640, 480, 16, 0);

if (screen == NULL)
{
   fprintf(stderr, "Error: Can't open window!  %s\n",
         SDL_GetError());
   exit(1);
}
```

Load Images



```
SDL_Surface * chicken_image_left[2];
SDL_Surface * chicken_image_right[2];

chicken_image_left[0]  = IMG_Load("chickenL1.png");
chicken_image_right[1] = IMG_Load("chickenL2.png");

chicken_image_left[0]  = IMG_Load("chickenR1.png");
chicken_image_right[1] = IMG_Load("chickenR2.png");
```

Again with the missing error checks!

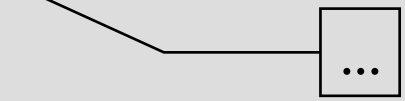


("SDL_image" is a helper library for SDL, also released under the LGPL. It supports GIF, JPEG, PNG and other formats.)

Event-driven programming

User presses a key

User holds the key down



User lets go of the key

Use flags to keep track of keys you care about
(like arrow keys and “fire” buttons)

Typical event loop

```
done = FALSE;

do
{
  while (events pending)
  {
     ... handle events...;
  }

  move objects, handle collisions & other game logic;

  draw the screen;
}
while (done == FALSE);
```

In many cases, you'll also want to 'throttle' the loop speed by idling at the end, if needed (e.g., to not go faster than, say, 60fps)

Chicken Variables

NOTE: Being a quick-and-dirty designer, I came up with these various variables as they became necessary. In other words, I designed the game while writing it. **BAD IDEA!** You can get away with it with little games like this, though. (But don't tell the profs I said that!)

```
int chicken_x;
int chicken_y;
int chicken_facing;
int anim_frame;
int chicken_hit_counter;
```

Where the chicken is on the screen

Direction chicken's facing (L/R)
Depends on last arrow key pressed

A little 'toggle' flag to switch between frames of chicken animation
Only toggles when an arrow key is pressed

Int chicken_hit_counter;

When the chicken bumps into a car, she doesn't get squished like in “Frogger.”

She gets pushed down the screen (away from the goal), and the user is unable to control her for a few seconds.

So, within the main loop of the game, we can do...

We can also draw a different chicken graphic while chicken_hit_counter>0

```
if (bumped by car)
  chicken_hit_counter = 20;    /* for example */
                ...
if (chicken_hit_counter == 0)
  ... user controls work normally ...
else
{
  move chicken downwards;
  chicken_hit_counter = chicken_hit_counter - 1;
}
```

Chicken initialization!

```
/* center of the screen */
chicken_x = (640 - 32) / 2;

/* bottom of the screen */
chicken_y = (480 - 32);

/* arbitrary... */
chicken_facing = LEFT;

/* JUST AS IMPORTANT as _x and _y! */
chicken_hit_counter = 0;

/* arbitrary; 1'd do as well, cuz it just toggles */
anim_frame = 0;
```

To be modular, you could write it as:
`(screen->w - 32) / 2`

Better yet,
`(screen->w - chicken_image_left->w) / 2`
(assuming all chicken images are the same size)

Finally, on to the main event loop!

```
int done;                    /* Or "unsigned char" or... */
SDL_Event event;

...

done = FALSE;

do
{
  while (SDL_PollEvent(&event) > 0)
  {
    ... handle the events! ...
  }
}
while (!done);
```

Events we can handle: QUIT

```
if (event.type == SDL_QUIT)
{
  /* User clicked "Close" button on the window,
     process received a friendly 'KILL' signal... */

  done = TRUE;     /* Simple! :^) */
}
```

Events we can handle: Key press

```
SDLKey key;
...

if (event.type == SDL_KEYDOWN)
{
  /* Key was PRESSED */

  key = event.key.keysym.sym;
                /* See why I made my own variable? */

  if (key == SDLK_q || key == SDLK_ESCAPE)
  {
    /* [Q] or [Escape] key; quit as well! */

    done = TRUE;
  }
}
```

Keep track of arrow keys

```
int keypressed_up, keypressed_down,
    keypressed_left, keypressed_right;

keypressed_up =    FALSE;
keypressed_down =  FALSE;
keypressed_left =  FALSE;
keypressed_right = FALSE;

                   . . .
else if (key == SDLK_UP)
  keypressed_up = TRUE;
else if (key == SDLK_DOWN)
  keypressed_down = TRUE;
else if (key == SDLK_LEFT)
  keypressed_left = TRUE;
else if (key == SDLK_RIGHT)
  keypressed_right = TRUE;
```

Tedious, isn't it!?
Why not use an array?

Notice that SDL defines arrows like so:

```
SDLK_UP    = 273,
SDLK_DOWN  = 274,
SDLK_RIGHT = 275,
SDLK_LEFT  = 276,
```

Ref: "SDL_keysym.h" header file

...and arrow key releases!

```
else if (event.type == SDL_KEYUP)
{
  /* A key has been RELEASED! */
  key = event.key.keysym.sym;

  if (key == SDLK_UP)
    keypressed_up = FALSE;
  else if (key == SDLK_UP)
    keypressed_down = FALSE;
  else if (key == SDLK_RIGHT)
    keypressed_right = FALSE;
  else if (key == SDLK_LEFT)
    keypressed_left = FALSE;
}
```

Look familiar?

Move the chicken!

```
if (keypressed_up)
{
  chicken_y = chicken_y - 4;
}
else if (keypressed_down)
{
  chicken_y = chicken_y + 4;
}

if (keypressed_left)
{
  chicken_x = chicken_x - 4;
}
else if (keypressed_right)
{
  chicken_x = chicken_x + 4;
}
```

(short for "keypressed_up == TRUE)"

Notice the use of "if" and not "else if" here! This allows for moving *diagonally* by holding two arrow keys at once!

Tests should occur to make sure chicken remains in bounds, too!
e.g.,

```
if (chicken_x < 0)
  chicken_x = 0;
```

Draw the screen

First step is to ERASE it. Cheap way:

Fill a rectangle with a solid color on the main window ("screen")'s backbuffer

1. Erase the 'backbuffer'

```
SDL_FillRect(screen,
             NULL,
```

Do it to the ENTIRE surface (0,0) to (screen->w – 1, screen->h – 1) *(More on this later)*

```
SDL_MapRGB(screen->format,
           128, 128, 128));
```

Determine the pixel value for the surface ("screen") that makes up the following color

Red = 128
Green = 128
Blue = 128
...
GREY!

2. Draw everything

[next slide]

3. Copy the backbuffer to the screen

```
SDL_Flip(screen);
```

Draw the chicken

Let's start by just drawing the same shape, no matter what...

```
SDL_Rect dest;

. . .

dest.x = chicken_x;
dest.y = chicken_y;

SDL_BlitSurface(chicken_image_hurt, NULL,
                screen, &dest);
```

SDL_Rect (pointer) describing where inside source surface to pull from
In our case, we want it ALL, so we can use "NULL" like we did with SDL_FillRect

Source surface

Destination surface

SDL_Rect (pointer) describing where in the destination surface to put it

Speed problems

If we ran that, it'd go as FAST AS POSSIBLE. The faster the computer, the faster it would wrong. *Typically, you don't want that.*

One solution is to alter the distance which objects move based on the calculated speed of the event loop.

Pro: Great for accuracy in 3D simulations & shooters
Cons: Lots of math, floating point required, not very *basic*

So instead, just assume a minimum requirement for the game, and then "throttle" it so it doesn't go faster than the FPS you declare.

(It certainly might go slower!)

Basic Throttle Technique

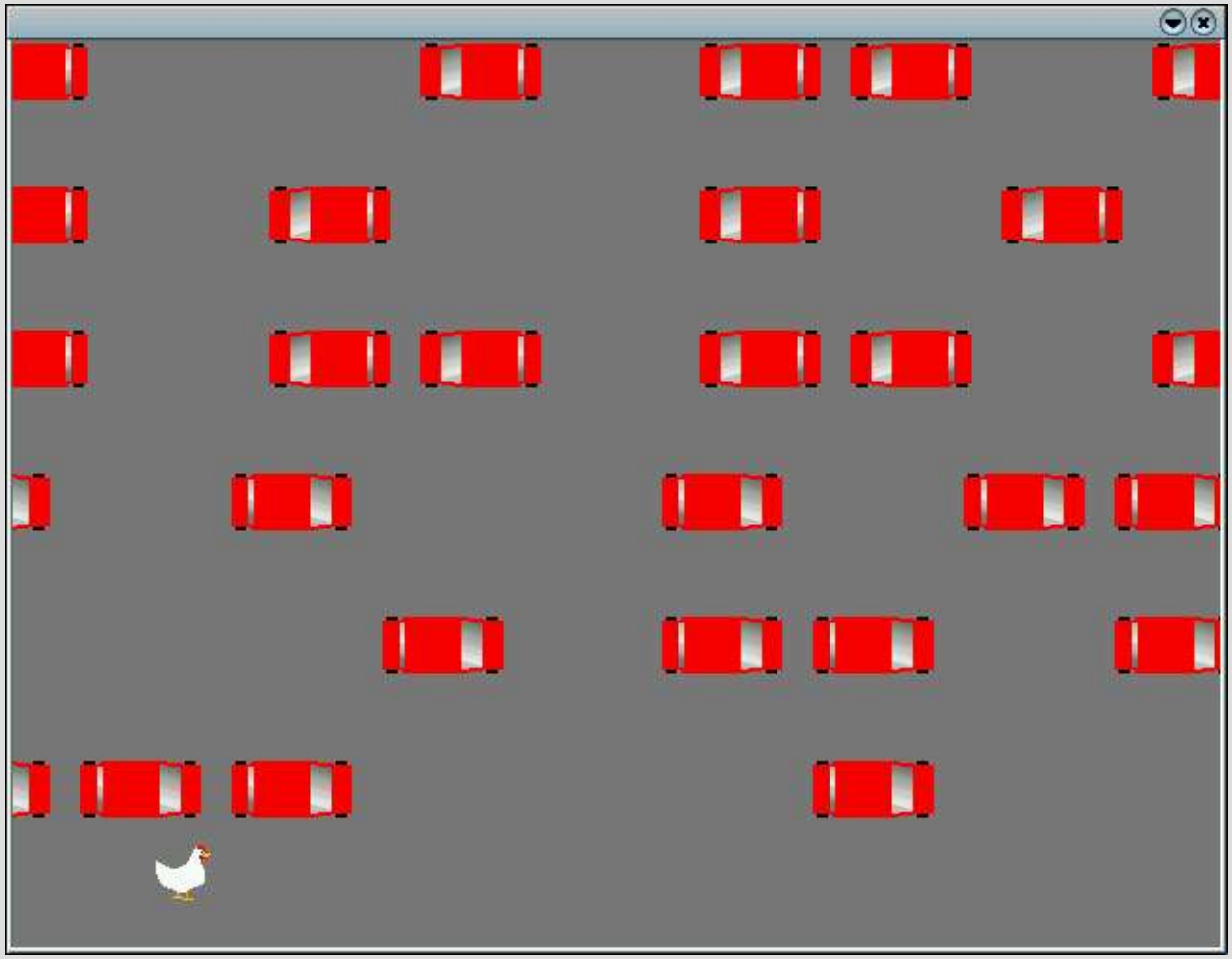
Our basic game loop:

```
do
{
   What time is it now?
   ... handle events ...
   ... game logic ...
   ... draw the screen ...
   Has it been 1/60th of a second yet?
   If not, pause the program until it has been
}
while (!done);
```

```
Uint32 last_time, cur_time;
                . . .
last_time = SDL_GetTicks();
```

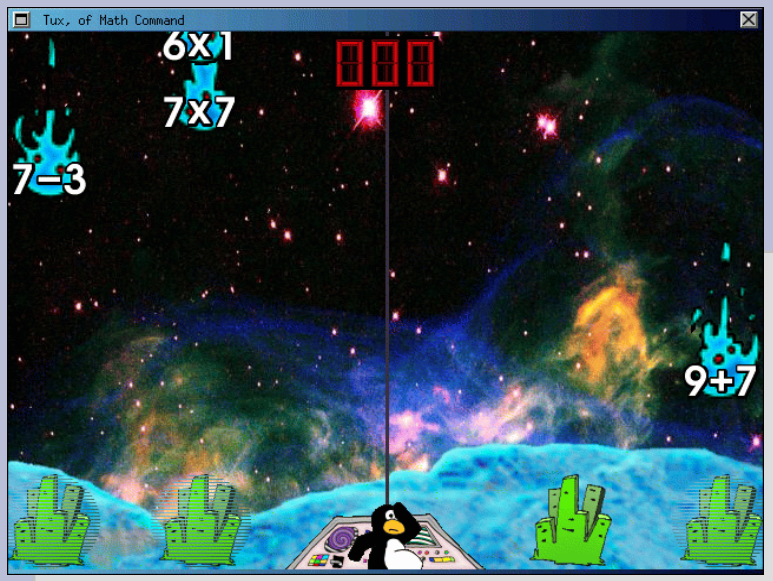
```
cur_time = SDL_GetTicks();

if (cur_time < last_time + (1000 / 60))
  /* Wait for the remainder */
  SDL_Delay(last_time + (1000 / 60) - cur_time);
```

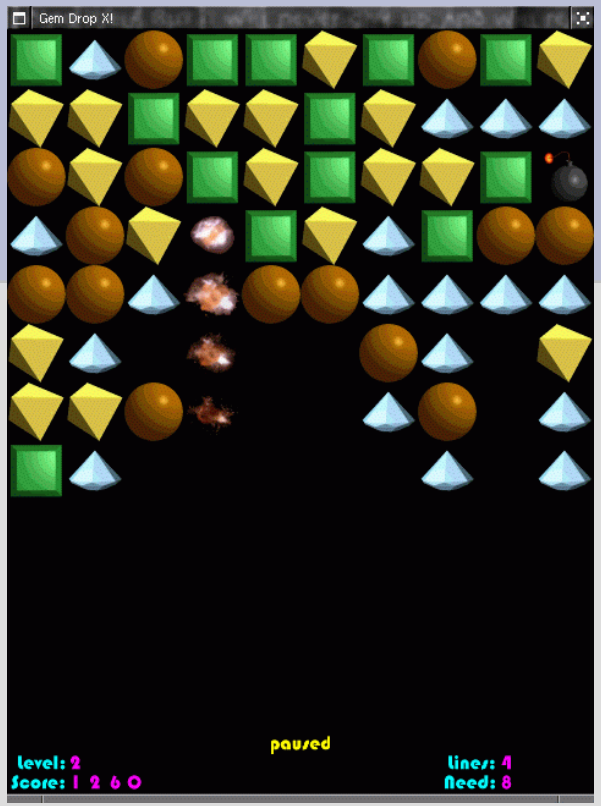
The game



Obvious improvements

- Different car colors
- Different kinds of vehicles (trucks, cycles)
- Lane markings and other artwork
- Varying traffic speeds
- Score display
- Timer
- Multiplayer
- Joystick control
- Difficulty options
  - Get knocked to the beginning
  - Inability to move left/right


Tux, of Math Command
6x1
7x7
7-3
9+7


Gem Drop X!
paused
level: 2
Score: 1260
lines: 4
Need: 8


Circus Linux!


0:23


DEFENDGUIN
press a key


BoboBot

References

New Breed Software
http://www.newbreedsoftware.com/

Simple DirectMedia Layer
http://www.libsdl.org/

Free Software Foundation *(GNU License info.)*
http://www.fsf.org/

Open Source Initiative *(general Open Source info.)*
http://www.opensource.org/

Linux Users' Group of Davis
http://www.lugod.org/