Introducing Computer Programming via Gameboy Advance Homebrew

Gary Kacmarcik
Google (Seattle, WA)

Sylvie Giral Kacmarcik
Whole Earth Montessori School (Bothell, WA)

Introduction

Introduce young students to computer science and engineering

Target 5th - 8th grade students

- After appropriate level of abstract thinking
  - Programming requires abstraction
- Before influence of social stereotypes
  - This can impact student's desire to learn

But try to appeal to a much wider audience

Goals

How computers work

- Demystify computers
- Connection between software/hardware

How to program

- Get comfortable with idea of programming
- Ignite spark of interest

Our approach

Motivate students with compelling goal

- Create GBA/NDS game, runs on real device
- “Homebrew” development

Provide real programming environment

- No sandbox or limitations

Connect software with underlying hardware

- Provide context for programming

Enhance “ownership” of programming project

Compelling goal

"Would you like to learn how to create your own Gameboy Advance/Nintendo DS game?"

Highly motivating for broad range of students

- From pre-K to college (and beyond)

GBA/NDS is first real electronic device

- Not a kid's toy

vs

Complete environment

No restrictions on what you can do

- “Keys to the kingdom”

Important for teenagers

- Easy to recognize playground environment
  - Can be de-motivating
- Sensitive to situation where they are not treated as an adult

Connect software and hardware

GBA is relatively simple

- No OS or VM between program and device
- Manipulate hardware registers

NDS is slightly more complex

- GBA + additional hardware (touchscreen)

Easy to make connection to hardware

- References made throughout class

Enhance ownership

Strongly believe that students should create all their own graphics

- Important for motivation
- Increased sense of ownership/accomplishment
- Implies that we need to start with 2D

Compare with 3D programming worlds:

- Students forced to rely on pre-generated models

What is “homebrew” software?

Homebrew software is:

- Written for proprietary hardware systems
  - Not typically programmable by end-users
  - Usually requires official devkit ($$$)
- Created by non-professionals (end-users)
  - “Hobbyist” programmers

Homebrew community

Requires:

- Development tools to be created
- The system to be reverse-engineered
- Homebrew community for each system

Tools made available to the community

- For Free

All major systems have a homebrew community

- With varying degrees of success

GBA/NDS homebrew

Mature homebrew community:

- Development tools:
  - devkitPro (devkitARM for GBA/NDS)
  - Various text editors/IDEs
  - Various graphic editing tools

- Emulators:
  - GBA: VisualBoyAdvance, no$gba
  - NDS: no$gba, DeSmuME, Dualis, iDeaS

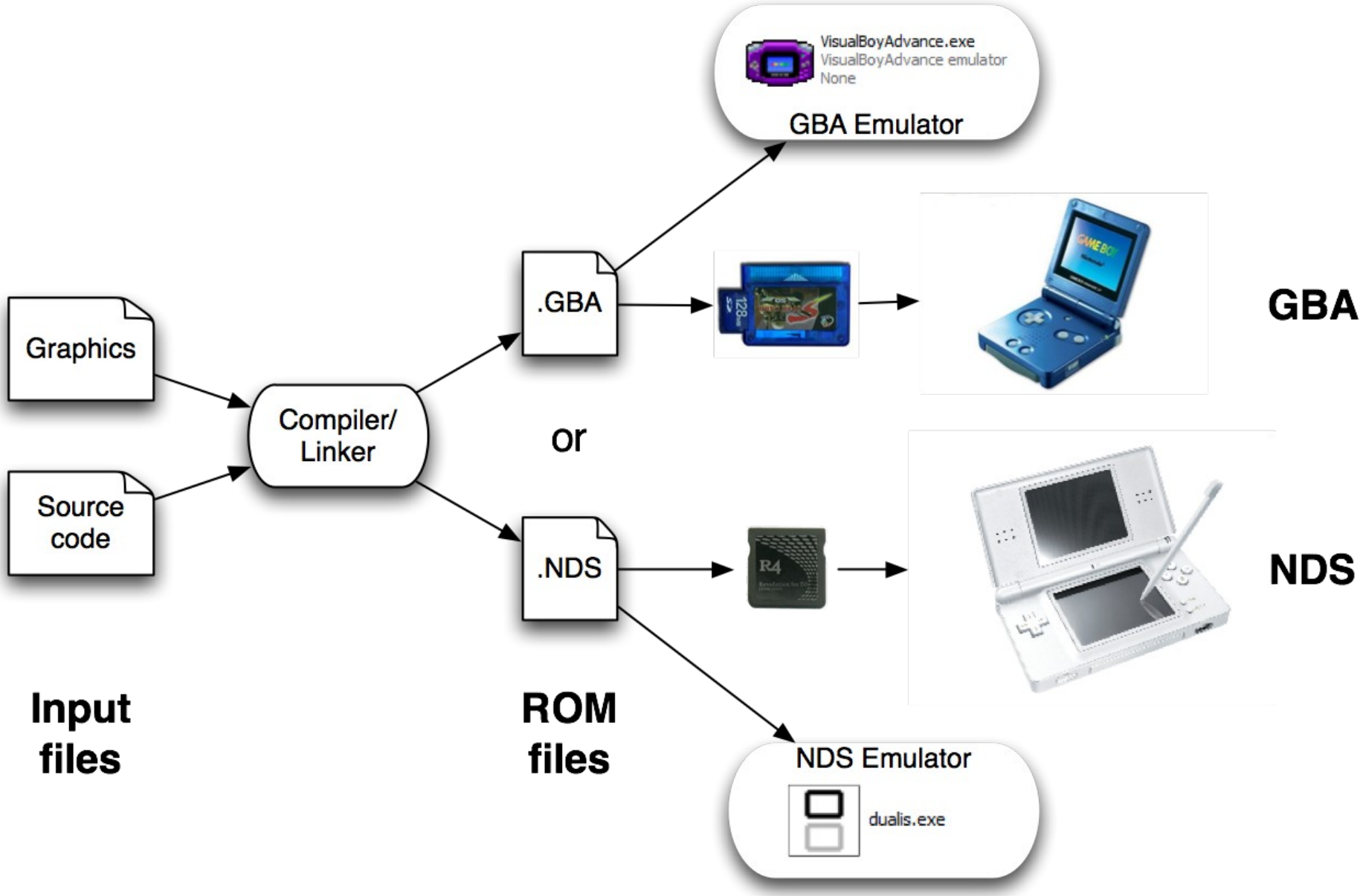
GBA/NDS cartridges

Run projects on real hardware

- GBA: SuperCard, MoviePlayer, ...

- NDS: R4, M3, DSTT, CycloDS, DSLinker, ...

Homebrew development flow



Homebrew caveats

Homebrew dev tools are not “friendly”

- Can be difficult to work with at first
  - Assume familiarity with command line
- Debugging environment is not ideal
- Not created with elementary students in mind

Once set up, however, it's straightforward

- With one exception:
  - Integrating graphics into your game

Graphic editing tools

Lots of 2D tile/map editors and conversion tools

- Mappy, Tiled, gfx2gba, …

Two broad categories:

- General purpose graphical tools
  - Need to select options to work on GBA/NDS
- Command line tools:
  - gfx2gba -D -fsrc -psprite.pal -t8 sprite.bmp
  - grit sprite.bmp -Mw 2 -Mh 4 -gB4 -pe 16 -U16 -ftc

Graphic processing

Problem:

- Need to import graphic files
- Process is error-prone

Solution:

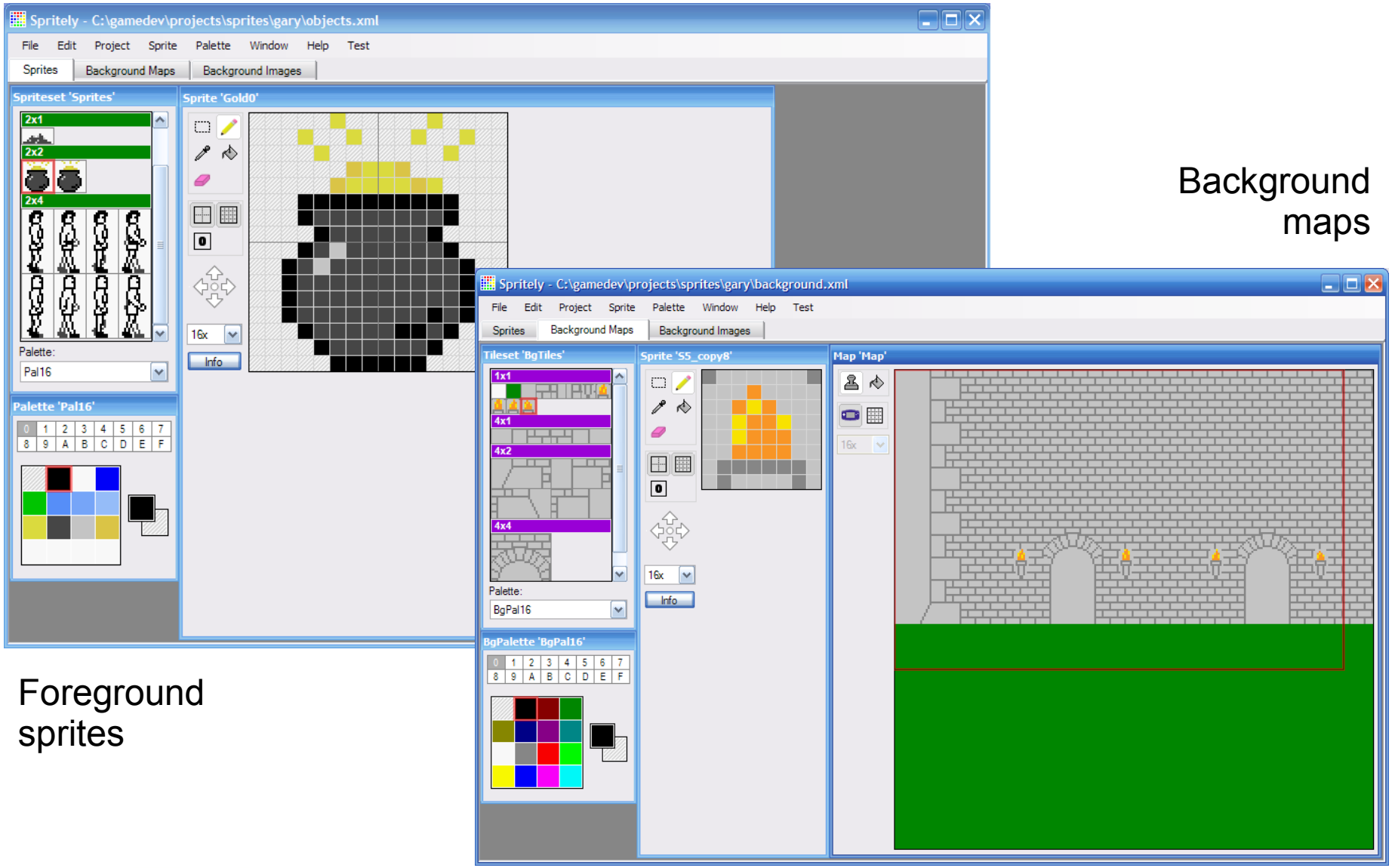
- Create tool specific for task

Spritely

Spritely is a tile/sprite/map editor

- Specifically for GBA/NDS
- Prioritize features for beginner game developers
- Students cannot make conversion mistakes

Spritely sprite/map editing



Background maps

Foreground sprites

Spritely Project Export

Can also export complete GBA/NDS project

- Starter project:
  - Draw > Export > Compile > Run
- Used as baseline for their own projects

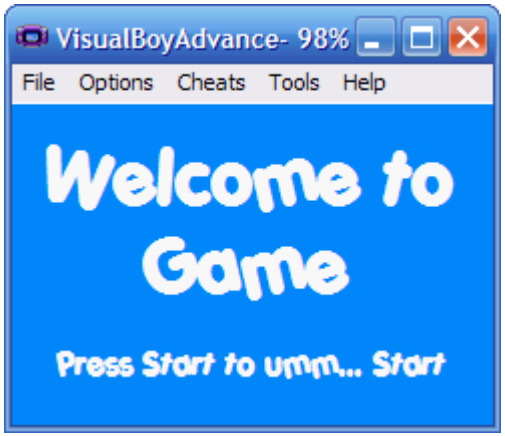
Spritely Demo

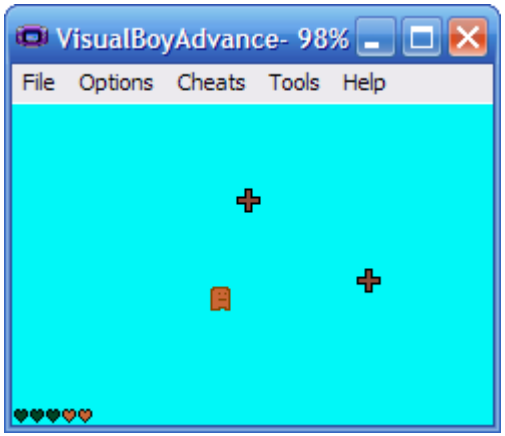
Spritely Tutorials

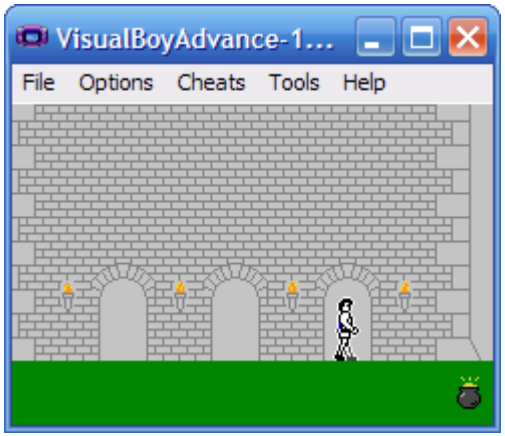
Programming structured as a series of tutorials:

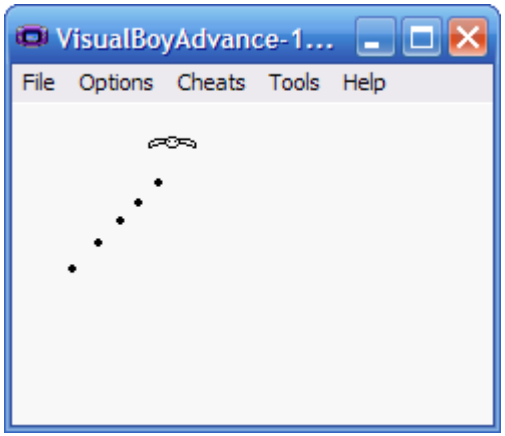
- Creating a ROM
- Creating and animating objects
- Collisions
- Projectiles & multiple projectiles (arrays)
- Gathering objects
- Levels (including title/game-over screens)
- Pong
- ...

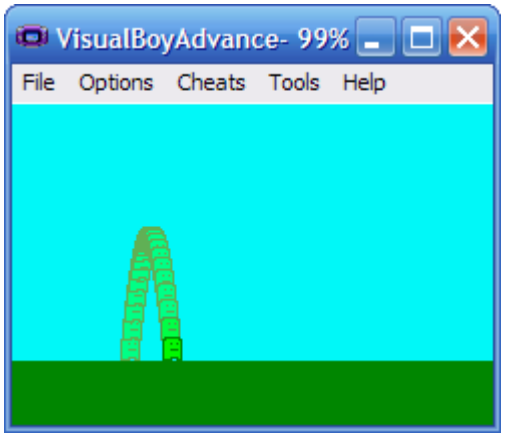
Sample tutorial projects

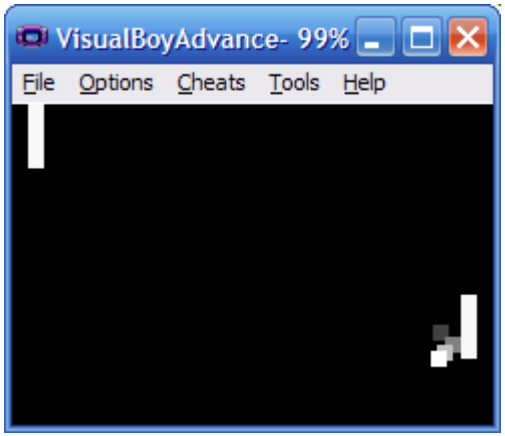




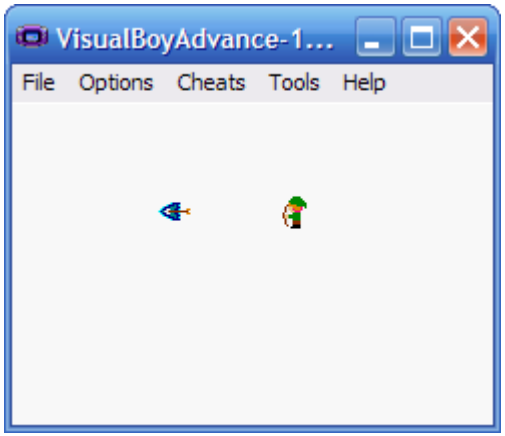


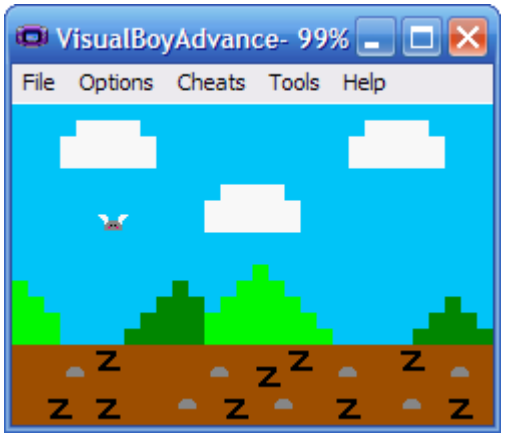


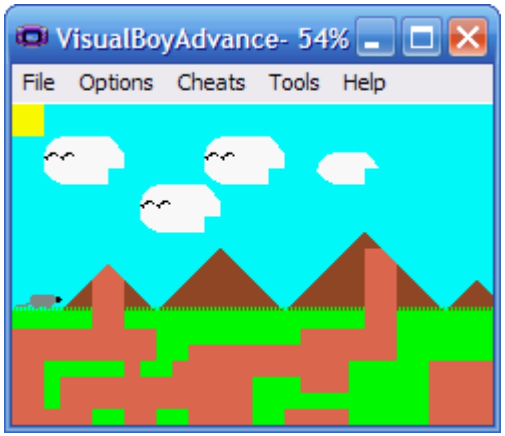


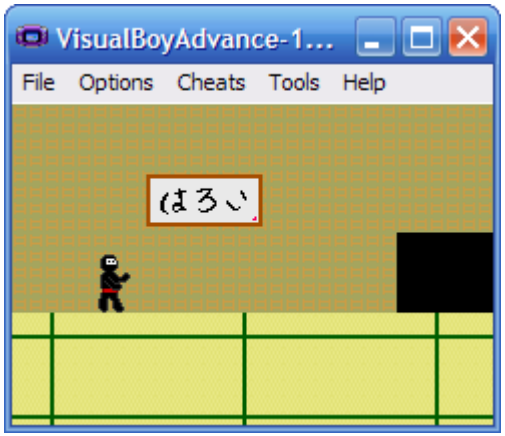


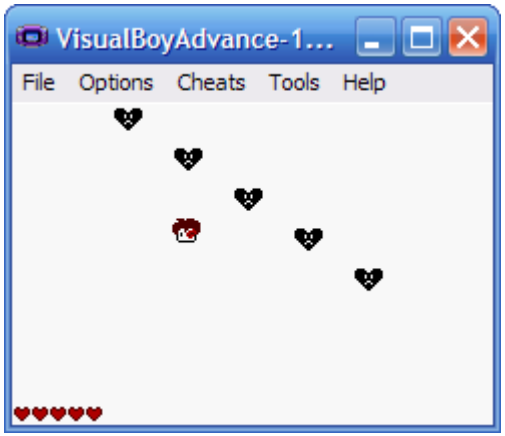
Sample student projects











Tutorial challenges

Two challenges with tutorials:

- Presenting code
  - Best way to present code edits in tutorial
- Keeping the tutorials up-to-date
  - Spritely is under development and changing

Presenting code

Students are unfamiliar with editing code

- Need to provide sufficient context

Custom Javascript pretty-printer to add annotations to code:

Step 5 : Adjust for size of player sprite

Make the following changes:

game_state.cpp — Lines 94 - 116:

```
        // If we need to move the player.
        if (dx != 0 || dy != 0) {
            // Calculate the player's new location.
            int x = _xPlayer + dx;
            int y = _yPlayer + dy;

            // Get the width/height of the player.
            int width, height;
            GetObjectSize(kObj_Player, &width, &height);

            // Don't let the player go outside the screen boundaries.
            if (x < 0 || x > SCREEN_WIDTH - width)
                dx = 0;
            if (y < 0 || y > SCREEN_HEIGHT - height)
                dy = 0;
```

Keeping tutorials up-to-date

Developing Spritely and tutorials simultaneously

- Feedback to improve program/tutorials
  - Restructure generated code
  - Add/remove base functionality

Don't break existing tutorials

- Need to constantly validate tutorials
- Easy with 1-2, challenging as you add more

Automated tutorial verification

Class organization

Class was offered as:

- A series of 1 hour classes after school
- ~32 weeks
- Small class size:
  - 8 students: 6 girls, 2 boys

Broad range of topics

Pre-programming skills

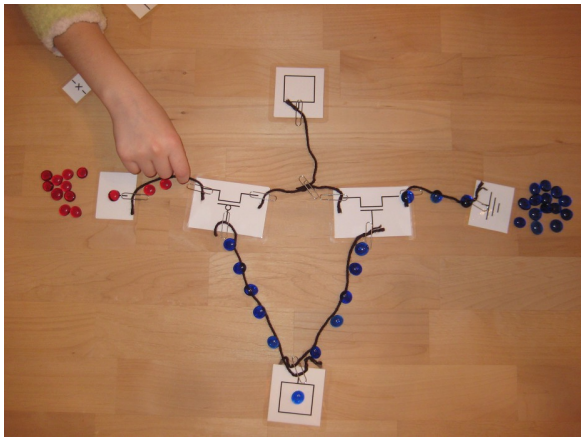
- Number systems, boolean logic

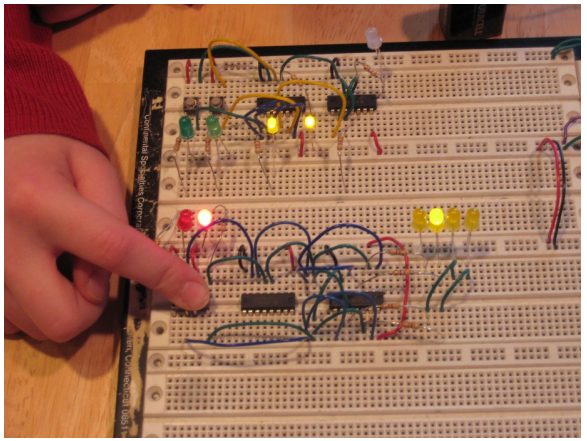
Digital hardware

- n/p-type MOSFETs, CMOS
- Hardware lab

Basic programming

- Variables, control flow





GBA/NDS Programming

GBA/NDS programming in 2nd half of class

References made to GBA/NDS throughout:

- Number systems
  - Draw 8x8 bitmaps and convert base 2 & 16
- Memory
  - Show how GBA carts map into upper address
- Hardware
  - Disassemble GBA

Programming Project

Lure/trick students into programming

- Start out with basic tutorials
- Continue by drawing sprites/maps for project
  - Students invest themselves in project
- Let student drive
  - “How can I...?” leads to related tutorial
  - Also peer driven “How did you do that?”

Evaluation

Goal is to spark interest in programming

- How do you measure that?

Un-prompted metrics:

- Observed in students without prompting
- Instead of asking if they would recommend the class, we observe whether or not they did

5 metrics:

- Drop, Recommend, Relate, Debug, Program

Evaluation

5 metrics:

- Drop – Did not complete class = 25%
- Recommend – Recommended class = 63%
- Relate – Related class info outside = 75%
- Debug – Independent debugging = 50%
- Program – Independent programming = 25%

Observed metrics, will tend to under-report

Conclusion

Overall:

- Successful in motivating students

But

- Approach not appropriate for all situations
- Teacher intensive, best with small class size
- Should probably follow visual programming:
  - Scratch, Alice, ...

We're releasing tools, tutorials & other materials

Questions?

Spritely and tutorials:

- http://code.google.com/p/spritely

Still under development

We welcome feedback/comments