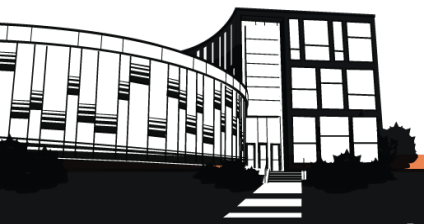
Developing Games on a limited platform

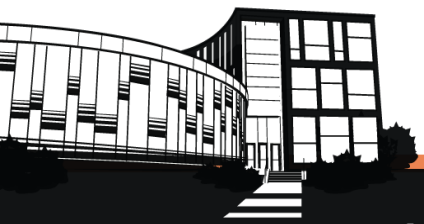
Game Engine
&
Art Implementation

Justin Lewis Kevin Hockey
Dave Silverman Scott JT Mengel



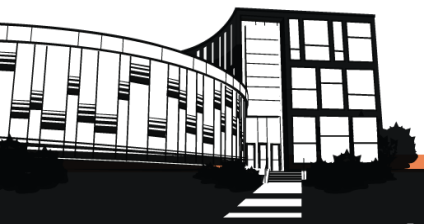
Overview

- The Platform
  - The One Laptop Per Child (OLPC) Project
  - The XO Laptop
- The Sugar Environment
  - The Operating System
  - Python Activities
- Fortune Hunter
- Game Engine
- Art Implementation
- Questions



The Platform

One Laptop per Child
XO Laptop



one laptop per child™

"Mission Statement: To create educational opportunities for the world's poorest children by providing each child with a rugged, low-cost, low-power, connected laptop with content and software designed for collaborative, joyful, self-empowered learning."

- Bridge the education gap in developing countries.
- "It's not a laptop project. It's an education project"

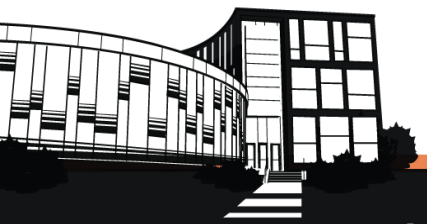
Quote Source: http://laptop.org/en/vision/index.shtml
http://laptop.org/en/vision/mission/index.shtml
Image Source: http://wiki.laptop.org/go/Category:OLPC_logos

The XO Laptop

- XO 1
  - CPU clock speed: 433 Mhz
  - Mass storage: 1024 MiB SLC NAND flash
  - DRAM memory: 256 MiB
- XO 1.5
  - CPU clock speed: 400 MHz to 1GHz, variable.
  - Mass storage: 4 GiB NAND flash in an internal microSD card
  - DRAM memory: 512 MB or 1 GiB

Hardware Stats: http://wiki.laptop.org/go/Hardware_specification_1.5
http://laptop.org/en/laptop/hardware/specs.shtml
Image Source: http://laptop.org/en/laptop/start/ebook.shtml

The Sugar Environment



sugar

"The award-winning Sugar Learning Platform promotes collaborative learning through Sugar Activities that encourage critical thinking, the heart of a quality education. Designed from the ground up especially for children, Sugar offers an alternative to traditional “office-desktop” software."

- Sugar is built on top of Fedora, a distribution of GNU/Linux, which is free software.
- It allows users to easily navigate a desktop environment, with activities instead of applications.

Image Source: http://wiki.sugarlabs.org/go/Marketing_Team/Logo
Quote Source: http://sugarlabs.org/

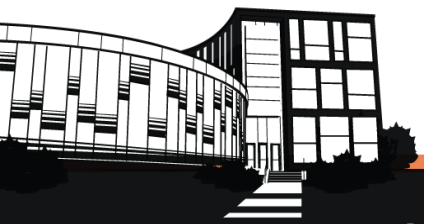
Activities

Activities are what Sugar Labs call applications that can run natively on their software and are designed to interface perfectly with their platform.

Activities are designed to download and install from the sugar browser with one click.

Mathematical Adventure: Fortune Hunter

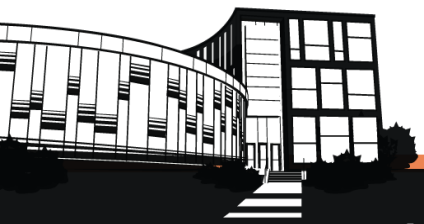
A game from
RIT's OLPC Course



Fortune Hunter

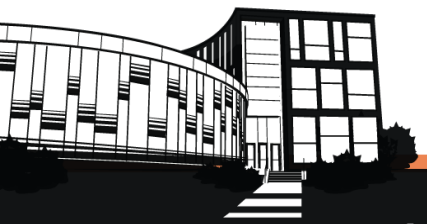
- A 4th Grade Dungeon Adventure styled Math Game
- Started in the Fall 2009 OLPC Course at RIT

Fortune Game Engine



Design of PyGame

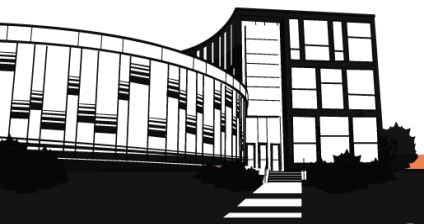
- Portable
  - Built on the Standard Drawing Library (SDL)
- Modular
- Simple
  - "Pygame is used in the OLPC project and has been taught in essay courses to young kids, and college students."
- Developer has control
  - "You control your main loop. You call pygame functions, they don't call your functions."



Quote Source: http://pygame.org/wiki/about

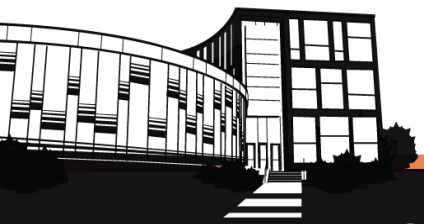
The Problem

- Developer has control
  - It is up to the developer to write clean code
  - Rapid Prototyping = less design
    - Large Main Loops
    - Leftover code
    - This was problematic in the original Fortune Hunter



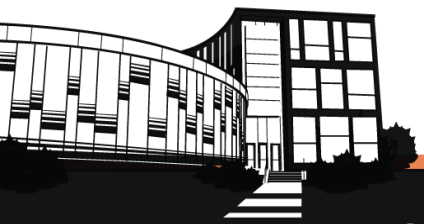
The Solution

- Fortune Engine
  - Forces developers to follow a design
  - Register for main loop functions: Draw, Event, and timers
    - Modular code
  - Register variables with game engine
    - Avoids Global Namespace
    - Provides debuging tools



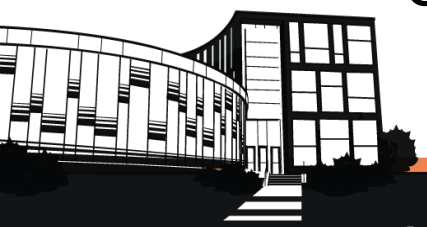
Game Engine Evolution

- First Design: Event driven loop.
  - Event must wake up draw loop
  - Timer event passed to all event handlers for animation
- Second Design: Threaded Event and Draw Loop
  - Animations run independent from event loop
  - Always drawing even when nothing is changing
  - Global locks to prevent changes during drawing
  - Standard threading is slow in python
- Final Design: Non-Blocking Event Loop
  - Animations can run independent of the event system
  - A No-Event will cause the draw system to run
    - No timers required to wake up the event loop



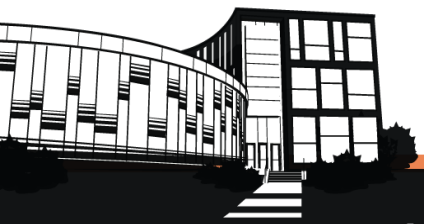
Game Engine Impacts

- Good for Open Source Environment
  - Forced to follow one coding design
    - Helps others pick up code
  - Modular design allows easier understanding
- Debugging
  - Developer Console
    - Inspection functions
    - Time Profiler
      - Which draw functions take up the most time?
      - Which event or timer is slowing the game down?
  - Simple main loop
    - Ideal spot to place debugging to statements to figure out causes of global lockups.

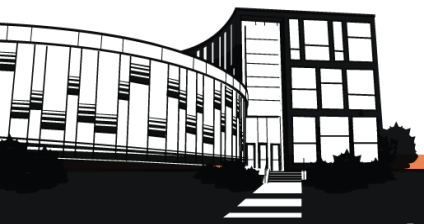


Game Engine Proof of Concept

- MAFH Game
  - Eliminated large amounts of code determining game state
  - Game Engine keeps track of enemies and user data
    - No objects in global memory space
  - Divided code into modules
  - Drawing is now modular
    - Before it was difficult to determine when and how things should be drawn
  - Runs in animation mode, always drawing when idle
- Lemonade Stand
  - Runs in non-animation mode, draws only when screen is “dirty”



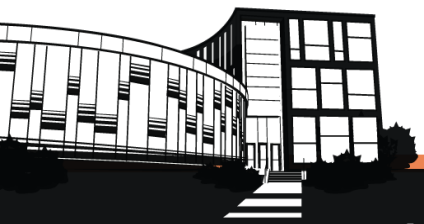
Art Implementation



Graphics in Programs

- For handling graphics we used the Python library, 'Pygame'
  - Based on the Standard Drawing Language which is built with ANSI C

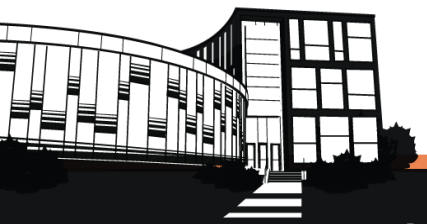
- The main graphics goal: research methods to improve program speed without sacrificing quality
  - Methods of loading in art assets
  - Handling art assets' meta data gracefully
  - Intelligently refreshing pixels that have changed since the last refresh



"Dirty Blitting"

- A common technique for refreshing the screen economically

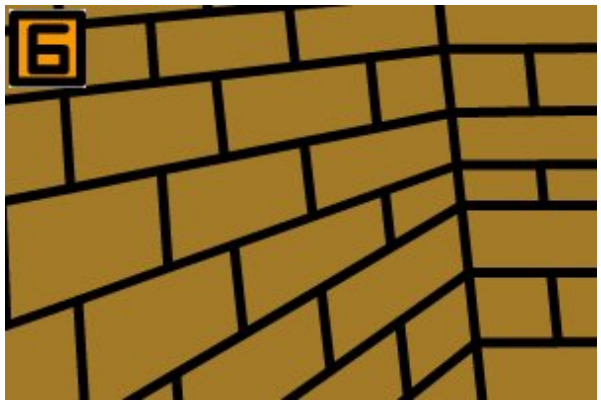
- What it means: Only refresh the pixels that have been marked as 'dirty', i.e. only refresh the pixels that have been marked as modified since the last refresh
  - Time is lost in tracking and calculating which pixels to draw
  - More time is ultimately saved from drawing less pixels to the screen



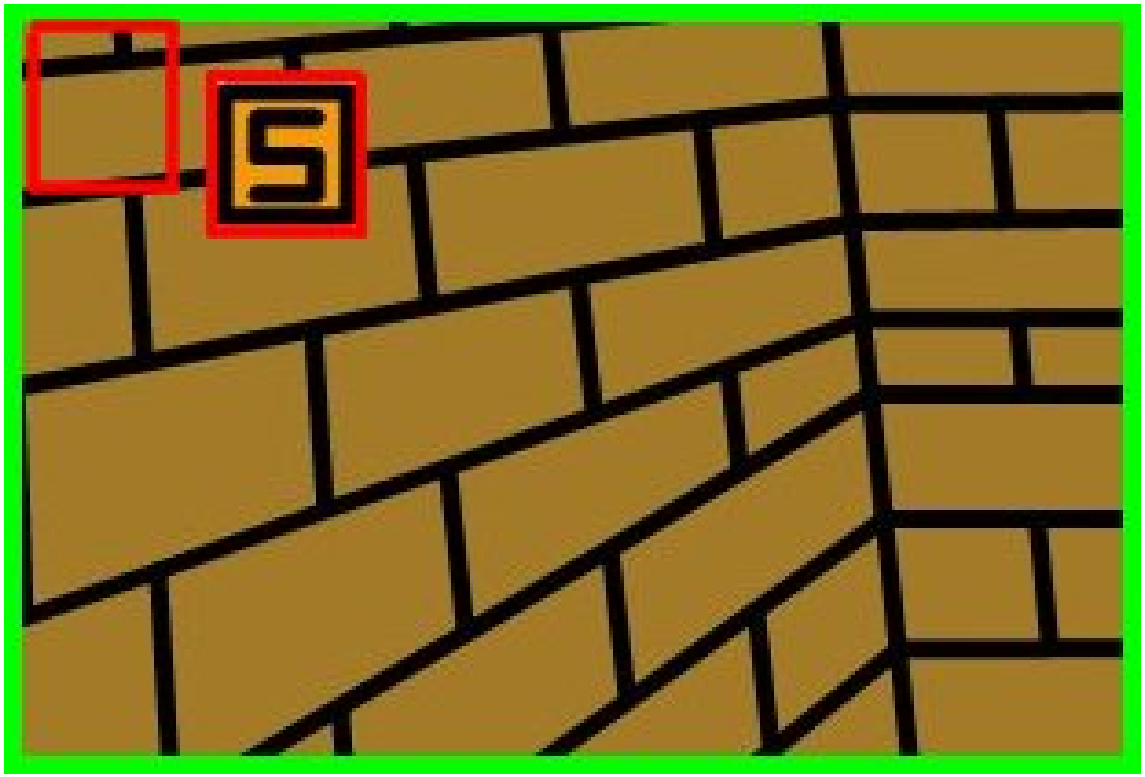
"Dirty Blitting"

Source: Screen capture from our test included in our publication.

Time 1



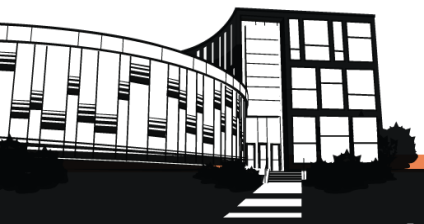
Time 2 (enlarged)



Red - Area refreshed using Dirty Blitting
Green - Area refreshed using original method

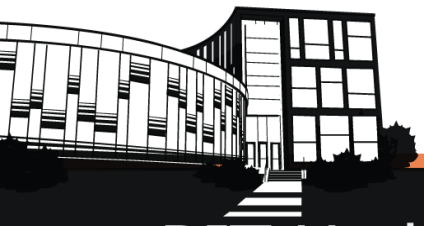
Loading Images

- Filetypes have a significant impact on performance and speed
- Even faster way: Explicitly converting the image upon loading
  - Converting turns file type formats into pixel type formats
  - Pixel format is the form of the image native to the program
  - Without converting, programs convert as the image is used



If you plan on using a tool multiple times in a short period, you would keep the tool out instead of returning it to your tool box.

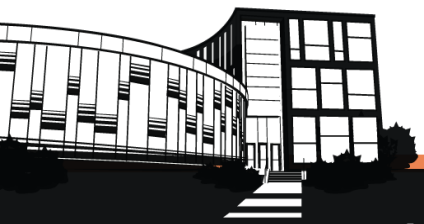
By explicitly converting our image to a pixel format once, we avoid the program automatically and redundantly converting the image (or 'retrieving it from the tool box') every time it's used.



Idea Behind System

Wanted a system that made animation easy for user. Some features we wanted to implement:

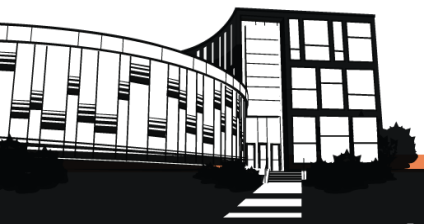
- Able to create characters with multiple animations
- Create menu items with multiple states/images
- Able to do all or most drawing for game/software



Using Sprites

When building our system we decided to build it on top of the pre-existing Sprite and SpriteGroup classes.

- Makes Dirty Blitting easier
- Already included a bunch of functionality we needed
- Not challenging to work with

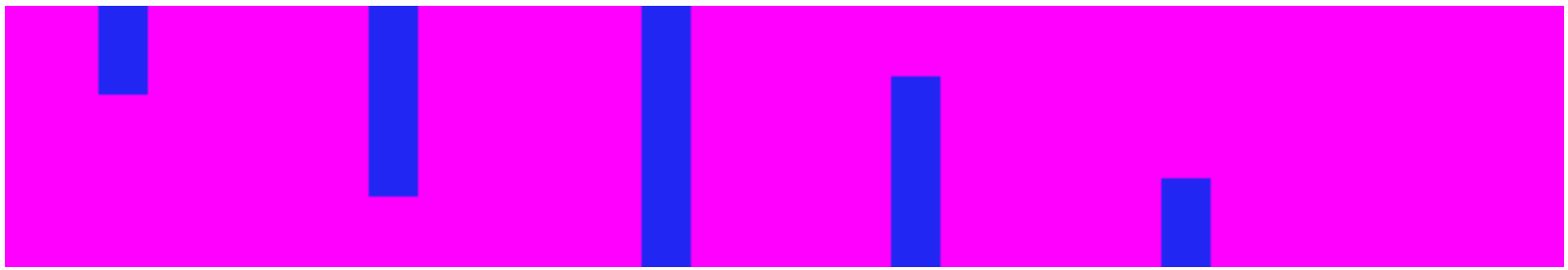


How System Works

We created four unique classes

- DrawableObject
  - DynamicDrawableObject
  - DrawableFontObject
- Scene

We also used a class we found on the internet called Spritesheet



Using the System

- Create a DrawableObject, of any type, for an object that you want to draw
- Add your DrawableObjects to your Scene
- Update the Scene, and thus also update all DrawableObjects that are being used
- Draw Scene to the Screen

Questions

http://foss.rit.edu/projects/fortunehunter

FOSS @ RIT