Building your Game
CMSC425.01 Fall 2019

Sit at the same table as last class
Administrivia

• Get started with Unity
  • Install Unity
  • Find references

• Project 1
  • Variation on Roll-A-Ball tutorial

• Today – Questions, rather than activities
Game systems this semester

**Processing**
- Interactive version of Java
- Used to illustrate concepts
- Not a game engine but has rich libraries
- [https://processing.org](https://processing.org)

**Unity**
- Full game engine
- Used for projects and assignments
- [https://unity3d.com](https://unity3d.com)
Today’s questions

How do you build a real time, interactive game?

What are the key elements of a game engine?
Game 1: Zork

- Early text based game
- Text of places and objects
- Simple command language
- Navigation by text

Q: Can we abstract and write a text game engine?
Game 1: Zork

- Early text based game
- Text of places and objects
- Simple command language
- Navigation by text

Q: Can we abstract and write text game engine?

Yes

Need:
- Code engine
- Command parser
- Text file descriptions
- Graph of locations
- User item bag
- Read/parse/do loop

**West of House**

ZORK I: The Great Underground Empire
Copyright © 1981, 1982, 1983 Infocom, Inc. All rights reserved.
ZORK is a registered trademark of Infocom, Inc.
Revision 88 / Serial number 840726

You are standing in an open field west of a white house, with a bearded front door.
There is a small mailbox here.

>open mailbox
Opening the small mailbox reveals a leaflet.

>read leaflet
(Taken)
"WELCOME TO ZORK!

ZORK is a game of adventure, danger, and low cunning. In it you will explore some of the most amazing territory ever seen by mortals. No computer should be without one!"
Game 1: Zork

- "Interactive fiction"
- Existing text engines:
  - Adrift, Inform, Quest

- Why care?

- Emphasis on story and language, not glitz
- The skeleton of a game
Game 2: Pong!

- Flatworm of interactive games
- Simple, but complete interactive game
- Example in Processing
Game 2: Pong!

• **Q**: How would you code this?

• What elements needed?
Game 2: Pong!

- Basic game loop

Initialize

do
  update ball (physics)
  update paddle (user input)
  if (collide) do something
  draw stuff
until done

Clean up
Game 2: Pong!

- Basic game loop

  Initialize
  do
    update ball (physics)
    update paddle (user input)
    if (collide) do something
  draw stuff
  until done
  Clean up

- Update ball

  - Very simple physics
    \[ x += dx; \]
    \[ y += dy; \]

  - Can add acceleration
    \[ dx += ddx; \]

https://processing.org/examples/bouncingball.html
Game 2: Pong!

• Basic game loop
  Initialize
do
    update ball (physics)
    update paddle (user input)
    if (collide) do something
  draw stuff
until done
Clean up

• Update paddle
  • Poll device – interrogate
    if (keyPressed && keyCode == DOWN)
      py = constrain(py+2,0,height);
Two form of user/system input

- **Poll device**
  - Initiate in your code
  - Read fixed memory location updated by system

- **Event driven**
  - Initiated by system
  - Not under your control
  - You write `callback` routine to service event (or `event handler`)

```c
void mousePressed()
{
    save("image.jpg");
}
```
void setup() {
    size(400,400);
}
void draw() {
}
void mousePressed() {
    ellipse(mouseX, mouseY, 20, 20);
}
void keyPressed() {
    save("pic.jpg");
}
Game 2: Pong!

• Basic game loop

  Initialize
  do
    update ball (physics)
    update paddle (user input)
    if (collide) do something
  draw stuff
  until done

  Clean up

• if (collide) do something
  • If hit wall or paddle, take action

if (pong.hitLeft()) {
  pong.reverseX();
}

Game 2: Pong!

• Basic game loop
  Initialize
  do
    update ball (physics)
    update paddle (user input)
    if (collide) do something
  draw stuff
  until done
Clean up

• draw stuff
  • Draw the arena, paddle and ball
    // draw ball
    color c = color(255,0,0); // red(RGB)
    fill(c);
    ellipse(x,y,radius,radius);
Unity game loop

Initialize game
do
  Physics (+collision)
  Input
  **Game logic(new)**
  Rendering
  GUI rendering
loop
Clean up
Time!

• Frame time (not constant)
  • Things executed every frame
  • Most important is rendering of scene

• Physics time
  • Steps in physics simulation
  • May run faster than frame time to get physics right (avoid big steps)

• Real time
  • System clock
  • For syncing music, video, other things that need real time
Game 3: Asteroids!

• More objects
  • Ship
  • Bullets
  • Asteroids
  • Enemy ship
  • GUI: Score, remaining ships

• Q: How upgrade our Pong game?
Game 3: Asteroids!

• Big change: more objects
  • Ship
  • Bullets
  • Asteroids
  • Enemy ship
  • GUI: Score, remaining ships

• Q: How upgrade our Pong game?
  • Object list

• List of game objects
• In loop
  • Update all
  • Interact! (time expensive)
  • Render all
Game 3: Asteroids!

- More objects
  - Ship
  - Bullets
  - Asteroids
  - Enemy ship
  - GUI: Score, remaining ships

Q: How upgrade our Pong game?
  - Object list

Object hierarchy

- Q: How design inheritance hierarchy for Asteroid game objects?
Unity – not OOP, but Entity-Component

- More like interfaces in Java

- Bullet
  - Implements Draw (Bullet shape)
  - Implements BallasticMotion
  - Owns Collider component

- Asteroid
  - Implements Draw (Asteroid shape)
  - Implements BallasticMotion
  - Owns Collider component

- Ship
  - Implements Draw (Ship shape)
  - Implements UserControlledMotion
  - Owns Collider component
  - Owns Shoot component

- Score
  - Implements Draw (Score shape)
  - No collider component, no motion
Scene graph vs. Object list

Object list in Asteroids
• All objects are simple, no articulated motion

Scene graph
• Directed graph, compound objects
• May share subparts
• Subparts have own displacements
Model View (MV) and rendering

- **Model of object stored**
  - Circle: \( (x, y, r, \text{color}) \)
    - Location \( x, y \)
    - Radius \( r \)

- **View of object rendered**
Model View (MV) and rendering

- **Model** of object stored
  - In 3D
    - Store list of vertices and polygons

- **View** of object rendered
  - Render object in 3D (using GPU)
Independence of model and view

- **Can render 3D model**
- From different viewpoints
  - Eg, split screen simultaneously
  - Change of perspective
- In different ways
- At different levels of detail
  - (far objects, less LOD)
Not all game objects are rendered (visible)

• Cameras/lights can move & behave but aren't rendered in game
• Model, no view except in mock up
Model View Controller (MVC) program

• Multi-user game

• Shared Model/Database

• Different Views

• Coordinated controllers

• (BTW – this could be Accounting system, any multi-user app)
Rendering!

• Convert 3D polygonal model to 2D image

• Do it well
• Do it cheaply
• Do it fast

• How?
Rendering

• Step 1: Elements of model

• Geometry: polygonal mesh
  • 3D points
  • Topology (graph structure)

• Appearance: color
  • Texture
  • Procedural shader

• Articulation/motions
Rendering

• Step 2: Scene elements

• Figures plus

• Camera

• Lights

• Skybox
Rendering

• Step 3: software graphics pipeline
• In 3D compute interaction between lights, model camera (math!)
• In 2D do low level rendering to display triangles with color
Rendering

• Step 4: hardware pipeline

• Push (immense data) to GPU

• Use dedicated bus (north bridge)

• Use GPU memory to pre-load textures, models, send only recent motion data
Summary

• After today you should be able:
  1) Explain the separation between game engine and game logic, assets
  2) Outline and explain a basic game loop and its stages
  3) Distinguish polling and events for user (network, game) input
  4) Read and explain an event driven real time graphics program
  5) Explain the different clocks used in a game
  6) Describe an object list, and a scene graph, for game objects
  7) Differentiate between OOP and Entity-Component systems
  8) Explain the elements of Model-View-Controller systems (MVC)
Putting it all together

- Key elements of typical game engine?
- Lots of parts to full system!
- Don't memorize diagram, but get high level view
  - Gameplay (high level game loop)
  - Model/scene graph+asset management
  - Physics/collision
  - Player/network interface
  - GUI
  - Rendering
Readings

• David Mount's lectures
• This class:
  • "Computer Game and Graphics System Architectures"
  • Next class:
  • "Intro to Unity"
• Pong code on web site – optional to read or run, but Processing is fun

• Other readings
  • Unity manual
  • Michael Kissner Gamasutra
Next: Moving on to Unity

• Will refine and explain these ideas through the semester

• You should
  • Install Unity
  • Do Roll-a-Ball tutorial
  • Start working on Project 1

• Ideas from today apply Unity