CMPM 120

Software Architecture
Objectives

By the end of today you should be able to...

1. **Software Architecture**
   a. Define software architecture
   b. Discuss how to apply software architecture when implementing a game

2. **Dialog Systems**
   a. Discuss how the requirements translate into software architecture
   b. Apply the process for figuring out the architecture of your own games

3. **Finite State Machines**
   a. Describe a Finite State Machine
   b. Implement a FSM in your own games
Revising Past Assignments

Since the point of the exercises is to measure your understanding of the material, I will allow you to submit a revision of your past assignment as long as:

➔ Your updated submission demonstrates your understanding of the material
  Include lots of comments, explaining why you chose to implement your solution in that way
  If I can't understand why you made your decisions, you don't get the points
➔ Late penalties still apply, but from the point of your original turn
  I want to encourage you to turn stuff in on time
  Turning stuff in late makes extra work for both of us
➔ Revision grading will happen at a time of my discretion
➔ No revisions will be accepted past the end of August 23rd
➔ **Does not apply to the final project:** the final project milestones are **hard deadlines**

Also: Tell me about it!
Problem Solving

**Define** the actual problem

**Think** about it

**Plan** a solution, including alternate plans

**Carry** out the plan

**Look Back**: verify you solved the original problem.
Software architecture refers to the fundamental structures of a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations.

https://en.wikipedia.org/wiki/Software_architecture
Software architecture refers to the art and science of designing and implementing software products. [...] The software architecture is analogous to the set of engineering drawings and diagrams for a building. [...] It is not advisable to begin “construction” without understanding the full scope of the engineering responsibility.

Introduction to Software Engineering, Richard F. Schmidt, Software Engineering
“Making a game combines everything that’s hard about building a bridge with everything that’s hard about composing an opera,” he said. “Games are basically operas made out of bridges.”

Frank Lantz, as quoted in "Master of the Game (Diploma Says So)"
Architecture for Dialog Systems?
What do we need in a dialog system?

Think about it, write it down
Nathan's Architecture: Design Requirements

Character portrait (with tweens)

"typing" text

Dialog box

Text is read from external file

Prompt and wait for user input

HOMER: Tell me, O muse, of that ingenious hero who travelled far and wide after he had sacked the famous town of Troy.
Nathan's Architecture

1. Create structured dialog data in JSON
2. Create and position dialog box sprite
3. Check to see if there are dialog lines remaining in current conversation
4. Check to see if there is a new speaker and tween them into view (and tween out previous speaker)
5. Construct dialog by adding speaker + line
6. Create a timer to “fire” dialog letter by letter
7. Lock input until all characters have printed
8. Increment; repeat
Some dialog systems

Yi's PhaserDialog
https://github.com/kthtes/PhaserDialog

April Grow's dialog system

120 student Tina Peng's timed dialog
Excerpts from April Grow's presentation on Data
Game Mechanics

- Move with arrows, R to reset
- Collision:
  - Player collides with a slime
  - Spacebar if something to say
  - Hit Spacebar: Show text
  - Hit Spacebar again: Text goes away
- Example includes:
  - Custom bitmap font
  - Looping & not-looping examples
  - Simultaneous utterances between player & slimes
  - Prerequisite previous dialogues
    - Also between different slimes
Our Data: The Dialogue Object

- Unique ID (used as prerequisite reference)
- Which dialogue agent is speaking (unique name, part of their prefab)
- Which other dialogue agent is being spoken to (unique name, part of their prefab)
- Which other dialogue object is required to have been read (or not read) (via Unique ID)
- Whether or not to repeat this utterance infinitely
- The text to be displayed
Code Overview

**Phaser Game**
- Load Font/Parse Data
- Feed Data to Dialogue Managers
- Collide Dialogue Agents

**Dialogue Objects**
- DialogueManager
  - With which other dialogue agent am I colliding?
  - Does this agent have something to say to that agent?
  - Are the preconditions for that utterance met?
  - Has the player acknowledged reading this utterance?

**main.js**

**slime.js & player.js**

**“Dialogue Agents”**
- NPCs
- Player

**dialogueManager.js**

**dialogue.json**

Phaser sprite prefabs
Clear Interfaces Between Segments of Code

Clear Game Dialogue Triggers

Consistent data structure

Plaser Game
- Input Event
- Collision Event

Dialogue Objects

Event context

Animate on events
Pass on events to manager

“Dialogue Agents”

NPCs  Player

Handles all logic about dialogue regardless of agent specifics

DialogueManager
- Prerequisite Checks
- Prompt Displays
- Text Displays
- Dialogue States

Event context
You May Want to Author or Visualize Your Data

- Simple version: CSV via Excel!
  - Many authors hate but understand this format.
  - Must be parsed and structured into other formats
  - Graphs!
  - Useful algorithms/macros

---

**How an accountant created an entire RPG inside an Excel spreadsheet**

*Arena.Xlsm* puts macros to work for levelling, battling, and collecting items.

---

A communiqué from the emperor, above, expresses interest at my formidable skill in killing bunnies and koalas with rocks.
You May Want to Author or Visualize Your Data

- Less Simple: JSON or XML (or other markup language)
  - More easily used by programs directly as it has more structure/context
  - Good at representing hierarchies!
  - AUTHORS’ BANE!!!
You May Want to Author or Visualize Your Data

- Tools that read and/or manipulate JSON or XML
  - Structured data with minimal post-production? Check!
    - Bonus if you can reuse save/loading code
  - A shiny interface that’s readable and usable? You hope so
    - Beating excel’s authoring environment is a low bar
- Many engines or game companies have their own
- Many people make small web tools too
Our Use Case

- What we need out of our data
  - Nodes
  - Links

- (CS 101: Vertices and Edges)

- One way to organize the data:
  - Nodes: Dialogue Objects
  - Links: Prereqs

- Alternative:
  - Nodes: Agents
  - Links: Dialogue Objects between agents
Some Tools

- The internet is full of free programs that help us visualize data
- Most of them work more for the statistical stuff before
- But we can bend them to our will because graphs are common structures

Here are some systems:

- [https://developers.google.com/chart/interactive/docs/gallery](https://developers.google.com/chart/interactive/docs/gallery) (Google’s)
- [https://philogb.github.io/jit/demos.html](https://philogb.github.io/jit/demos.html) (Infovis: gradients o.o)
- [https://d3js.org/](https://d3js.org/) (choice for this demo)
  - Beware differences in v3 and later versions
  - The demo is in the most recent version, v5
Other Narrative Tools

Yarn: https://github.com/InfiniteAmmoInc/Yarn

Javascript port of Yarn: https://github.com/jhayley/bondage.js/

Ink (from Inkle): https://www.inklestudios.com/ink/

Javascript port of Ink: https://github.com/y-lohse/inkjs
Yarn, Ink, Twine...

LONDON, 1872
Residence of Monsieur Phileas Fogg.

Monsieur Phileas Fogg returned home early from the Reform Club, and in a new-fangled steam-carriage, besides:

"I'm off," said he. "We are going around the world!"

"Around the world, Monsieur?"

"I was utterly astonished.

"Well, I must say I am quite surprised."

"I'm quite sure of it.

"Out of course!"

"We shall circumnavigate the globe within eighty days!" He was quite calm as he proposed this wild scheme. "We leave for Paris at 8:15. In an hour."

End of story
Q: What are the differences between the dialog demo we just saw and a narrative tool such as Yarn or Ink?
What could the implementation look like?

???
Games of Emergence versus Progression

The point of this paper is frightfully simple: That most computer games are the combination of two different ways of presenting the player with a challenge, one which I will term emergence (simple rules combining, leading to variation) and one of progression (serially introduced challenges).

Jesper Juul: "The Open and the Closed: Game of emergence and games of progression". http://www.jesperjuul.net/text/openandtheclosed.html
Emergence versus Progression, Generalized

Agents

Narrative Progression

Resources

Tech Trees

Character Upgrades

Simulation

Upgrade Progressions - even indirect progressions, such as Subnautica

Crafting Trees

Hypertext
Finite State Machines
Let's go back in time...

...to week 2.
States bundle up a series of methods that help get the program into and potentially out of a section of gameplay.
Boot

Menu

Pre-Game

Game

Game Over

High Score

(OS)
You can think of states like spaces on a game board...

...where your game piece can only be in one space at a time

This is also called a Finite State Machine (FSM)
"A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems."

from *Design Patterns*, by Gamma, Helm, Johnson, Vlissides

---

**Design Patterns**

Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice. -- Christopher Alexander

A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences. It also gives implementation hints and examples. The solution is a general arrangement of objects and classes that solve the problem. The solution is customized and implemented to solve the problem in a particular context. - DesignPatternsBook

Some topics that categorize DesignPatterns into the GangOfFour categories:

Given that patterns could be applied to many different disciplines, I would suggest that we talk about SoftwareDesignPatterns, to differentiate from ArchitecturalDesignPatterns or other kinds. Then the question is, are there any design patterns that work across specific disciplines? I doubt it, although there may be some "meta" patterns...
Wiki Wiki Web

This website and the software it runs on were created by WardCunningham for the PortlandPatternRepository. It is home to an InformalHistoryOfProgrammingIdeas as well as a large volume of material recording related discourses and collaboration between its readers.

The content is written by the users -- people like you and me. Anyone can change any page or create new pages. Read the TextFormattingRules to find out how, and then go to the WikiWikiSandbox to try it yourself. Please use the WikiWikiSandbox if you want to experiment with how editing works. If you make a page you don't want to keep, just replace its text with the word "delete".

This website is the first ever "wiki", where content can be edited by any person. All other wikis, including Wikipedia and Wikiquote and Wiktionary, are descended from it. Wikipedia is now the fifth most visited website according to Alexa (Rank).
Robert Nystrom wrote a book about software patterns in game development:

http://gameprogrammingpatterns.com/

(and it can be read free online)
Some of these may look familiar to you.
Q: Why are state machines useful?
Games have a lot of complex behaviors

Perhaps you’ve noticed a bit of a problem when we try to implement complex interactive behaviors in our characters.

Even a basic double jump gets tricky as we start to add boolean flags to account for various states the character might be in.
As character behaviors get increasingly more complex, our gnarly chain of if/else statements and boolean flags starts to break down.
A Simplified FSM

- **Ducking**: Press A
- **Standing**: Press B
- **Attacking**: Release
- **Walking**: Press A
- **Jumping**: Press A
- **Double Jumping**: Press A

- **Press B**: transitions between states
- **Press A**: transitions between states
- **Release**: transitions between states
In its purest form, an FSM has:

**States**, 
**Inputs**, 
And **Transitions**

Source: Robert Nystrom, *Game Programming Patterns*, “State”
https://gameprogrammingpatterns.com/state.html
There are a **fixed set of states** that the machine can be in. Our machine can **stand, walk, duck, jump, double jump, and attack**.
The machine can only be in one state at a time.

What can't our machine do?
A sequence on *inputs* or *events* is sent to the machine.
What inputs does this machine respond to?
Each state has a set of *transitions*. 

- **Standing**: Press A to jump, Press B to walk or attack. Release to return to Standing.
- **Ducking**: Press A to release, Press B to attack or standing. Release to return to Ducking.
- **Attacking**: Press B to release, Press A to release or jump. Release to return to Attacking.
- **Walking**: Press A to jump, Press B to attack or standing. Release to return to Walking.
- **Jumping**: Press A to release, Press B to attack or standing. Release to return to Jumping.
- **Double Jumping**: Press A to release, Press B to attack or jumping. Release to return to Double Jumping.
Let's make a simple FSM

solid → melt → liquid
liquid → condense → gas
gas → vaporize → solid

freeze → solid

Instead of reinventing the “machine,” let’s build on someone else’s work. (Namely, Joshua Shepard’s JS State Machine object)
// Adapted from Josh Shepard @ https://github.com/jcd-as/nadion/blob/master/src/statemachine.js
// Additional annotations by Johannes Spaulding and Nathan Altice

// State Machine constructor function
varStateMachine = function (states, receiver) {
    this.states = states; // JSON object that holds all states
    this.receiver = receiver; // the object that "receives" the state
    this.initialState = undefined; // the object's initial state
    this.indices = {}; // array used for fast lookup of events and states

    // initialize indices and find the initial state
    for (var i = 0; i < states.length; i++) {
        this.indices[this.states[i].name] = i;
        if (this.states[i].initial) {
            this.initialState = this.states[i];
        }
    }
    // warn if there's no initial state
    if (!this.initialState) {
        console.warn("State Machine has no initial state!");
    }
    // set current state to initial state
    this.currentState = this.initialState;
};

StateMachine.prototype = {
    consumeEvent: function (e) {
        if (this.currentState.events[e]) {
            this.currentState = this.states[this.indices[this.currentState.events[e]]];
        } else {
            console.warn("State Machine called with invalid event: " + e + " for current state: " + this.currentState.name + "," );
        }
    },
    // retrieve the name of the current state
    getState: function () {
        return this.currentState;
    },
    // reset the state machine to its initial state
    reset: function () {
        this.currentState = this.initialState;
    }
};
This FSM implementation requires that we pass our states as a JSON object. Each state has a name and event(s). One state is flagged as our init state.
Also note the fancy new way we're loading our helper script...
FSM Demo

solid → melt → liquid → condense → gas → vaporize → liquid → freeze → solid
Water Phase Diagram

Types of Ices

Let’s check out another state machine implementation, this time with animation.

(This example is by Adam Roth, who uses an adapted state machine originally implemented by David Hayes.)
```csharp
// states
StateMachine = newStateMachine(23);
StateMachine.SetCallbacks(StNormal, NormalUpdate, null, NormalBegin, NormalEnd);
StateMachine.SetCallbacks(StClimb, ClimbUpdate, null, ClimbBegin, ClimbEnd);
StateMachine.SetCallbacks(StDash, DashUpdate, DashCoroutine, DashBegin, DashEnd);
StateMachine.SetCallbacks(StSwim, SwimUpdate, null, SwimBegin, null);
StateMachine.SetCallbacks(StBoost, BoostUpdate, BoostCoroutine, BoostBegin, BoostEnd);
StateMachine.SetCallbacks(StRedDash, RedDashUpdate, RedDashCoroutine, RedDashBegin, RedDashEnd);
StateMachine.SetCallbacks(StHitSquash, HitSquashUpdate, null, HitSquashBegin, null);
StateMachine.SetCallbacks(StLaunch, LaunchUpdate, null, LaunchBegin, null);
StateMachine.SetCallbacks(StPickup, null, PickupCoroutine, null, null);
StateMachine.SetCallbacks(StDreamDash, DreamDashUpdate, null, DreamDashBegin, DreamDashEnd);
StateMachine.SetCallbacks(StSummitLaunch, SummitLaunchUpdate, null, SummitLaunchBegin, null);
StateMachine.SetCallbacks(StDummy, DummyUpdate, null, DummyBegin, null);
StateMachine.SetCallbacks(StIntroWalk, null, IntroWalkCoroutine, null, null);
StateMachine.SetCallbacks(StIntroJump, null, IntroJumpCoroutine, null, null);
StateMachine.SetCallbacks(StIntroRespawn, null, null, IntroRespawnBegin, IntroRespawnEnd);
StateMachine.SetCallbacks(StIntroWakeUp, null, IntroWakeUpCoroutine, null, null);
StateMachine.SetCallbacks(StTempleFall, TempleFallUpdate, TempleFallCoroutine);
StateMachine.SetCallbacks(StReflectionFall, ReflectionFallUpdate, ReflectionFallCoroutine, ReflectionFallBegin, ReflectionFallEnd);
StateMachine.SetCallbacks(StBirdDashTutorial, BirdDashTutorialUpdate, BirdDashTutorialCoroutine, BirdDashTutorialBegin, null);
StateMachine.SetCallbacks(StFrozen, FrozenUpdate, null, null, null);
StateMachine.SetCallbacks(StStarFly, StarFlyUpdate, StarFlyCoroutine, StarFlyBegin, StarFlyEnd);
StateMachine.SetCallbacks(StCassetteFly, CassetteFlyUpdate, CassetteFlyCoroutine, CassetteFlyBegin, CassetteFlyEnd);
StateMachine.SetCallbacks(StAttract, AttractUpdate, null, AttractBegin, AttractEnd);
Add(StateMachine);
```

5400+ lines of Player code for Celeste.
It's worth trying a several different state machine implementations to see what's best for your game.

(This one’s by Dave Stewart)
Lifecycle Events

In order to track or perform an action when a transition occurs, five general-purpose lifecycle events can be observed:

- `onBeforeTransition` - fired before any transition
- `onLeaveState` - fired when leaving any state
- `onTransition` - fired during any transition
- `onEnterState` - fired when entering any state
- `onAfterTransition` - fired after any transition

In addition to the general-purpose events, transitions can be observed using your specific transition and state names:

- `onBefore<TRANSITION>` - fired before a specific TRANSITION begins
- `onLeave<STATE>` - fired when leaving a specific STATE
- `onEnter<STATE>` - fired when entering a specific STATE
- `onAfter<TRANSITION>` - fired after a specific TRANSITION completes

For convenience, the 2 most useful events can be shortened:

- `on<TRANSITION>` - convenience shorthand for `onAfter<TRANSITION>`
- `on<STATE>` - convenience shorthand for `onEnter<STATE>`

Observing Lifecycle Events

Individual lifecycle events can be observed using an observer method:

```javascript
fsm.observe('onStep', function() {
    console.log('stepped');
});
```
How to Use State Machines to Control Behavior and Animations in Phaser

Suppose you’re building a platformer game, where the hero can walk, jump and attack. The hero can jump while standing or walking, but he can’t attack while jumping or walking. Also, the player can not jump twice in the same frame, and must wait 3 frames before being able to jump again. This can be achieved with state machines in Phaser.
What do you want to learn?
Potential Topics

- Cameras
- Particles
- P2 Physics
- Time & Timers
- Advanced Git
- State Machines
- Text and Fonts
- Animation and Tweens

- CSS (& other web dev stuff)
- Audio
- Scaling
- ...something you want to know about!
Write down 3 things your game needs

What do you not know?
What do we need to talk about more?
Potential Topics

- Cameras
- Particles
- P2 Physics
- Time & Timers
- Advanced Git
- State Machines
- Text and Fonts
- Animation and Tweens

- CSS (& other web dev stuff)
- Audio
- Scaling
- ...something you want to know about!
More Debugging Tips
Useful random debugging advice

1. When you find a problem, change something so that same problem can't happen again
   a. assert()
   b. Keep a debugging notebook
2. Make debug tools
   a. Quicker feedback is better
   b. Display values live if possible
3. Only make one change at a time and then test it
4. Just because you paused the game doesn't mean it's paused
   a. And stopping one update doesn't mean you stopped all of them
5. console.log() is slow
   a. Faster to print an array as a string than to individually print the contents
Useful random debugging advice

Walk through your code step by step, explaining to yourself what is supposed to happen
Useful random debugging advice

1. When you find a problem, change something so that same problem can't happen again
   a. assert()
   b. Keep a debugging notebook

2. Make debug tools
   a. Quicker feedback is better
   b. Display values live if possible

3. Only make one change at a time and then test it

4. Just because you paused the game doesn't mean it's paused
   a. And stopping one update doesn't mean you stopped all of them

5. console.log() is slow
   a. Faster to print an array as a string than to individually print the contents
Useful random debugging advice

1. When you find a problem, change something so that same problem can't happen again
   a. assert()
   b. Keep a debugging notebook

2. Make debug tools
   a. Quicker feedback is better
   b. Display values live if possible

3. Only make one change at a time and then test it

4. Just because you paused the game doesn't mean it's paused
   a. And stopping one update doesn't mean you stopped all of them

5. console.log() is slow
   a. Faster to print an array as a string than to individually print the contents
Useful random debugging advice

1. When you find a problem, change something so that same problem can't happen again
   a. assert()
   b. Keep a debugging notebook

2. Make debug tools
   a. Quicker feedback is better
   b. Display values live if possible

3. Only make one change at a time and then test it

4. Just because you paused the game doesn't mean it's paused
   a. And stopping one update doesn't mean you stopped all of them

5. console.log() is slow
   a. Faster to print an array as a string than to individually print the contents
Useful random debugging advice

1. When you find a problem, change something so that same problem can't happen again
   a. assert()
   b. Keep a debugging notebook

2. Make debug tools
   a. Quicker feedback is better
   b. Display values live if possible

3. Only make one change at a time and then test it

4. Just because you paused the game doesn't mean it's paused
   a. And stopping one update doesn't mean you stopped all of them

5. console.log() is slow
   a. Faster to print an array as a string than to individually print the contents
AABB characters and slopes

An example of a real-world physics-and-debugging problem in a game with 2D physics like yours

https://twitter.com/eevee/status/1133248372624613376