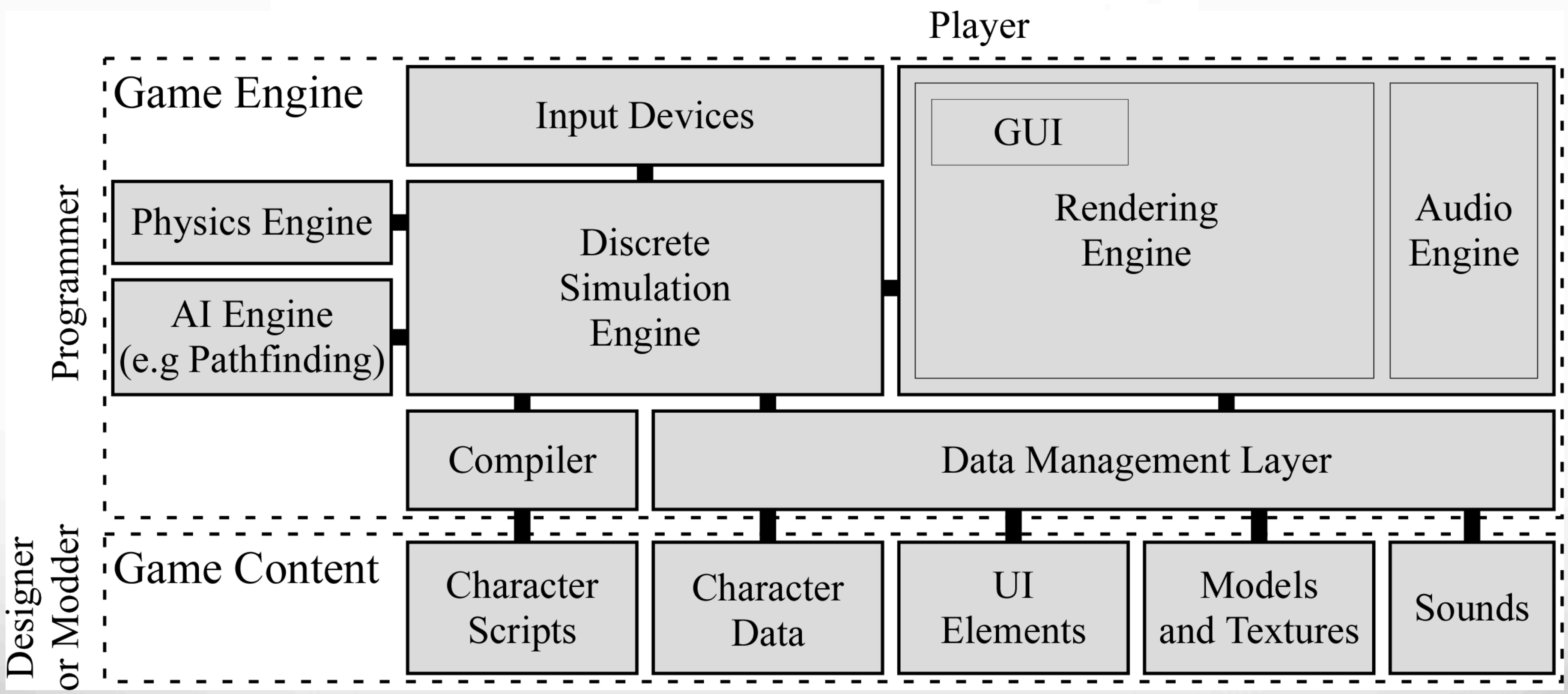
Lecture 10:

Game Architecture

Architecture: The Big Picture

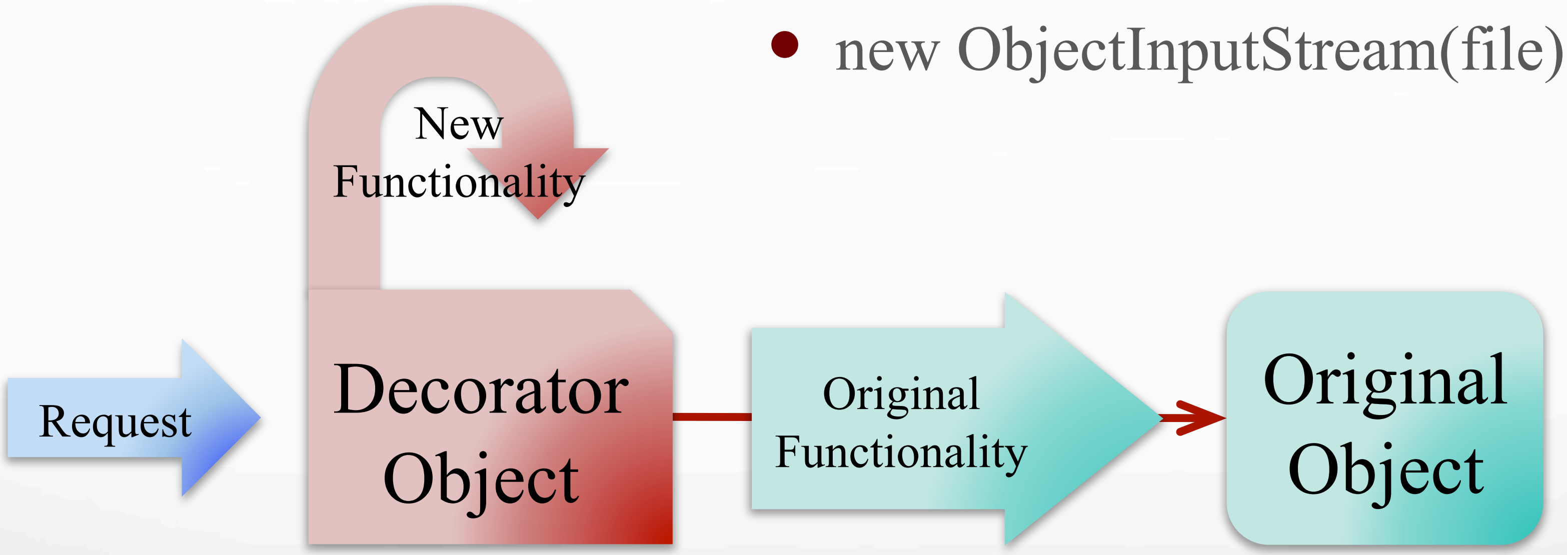


2110 Supplemental: Software Patterns

- Reusable solution to a reoccurring problem
  - Template, not a single program
  - Tells you how to design your code
- Useful for dynamic functionality
  - Object starts life as one class
  - Cannot "change its class" later
  - How do we add new functionality?
  - **Example**: Java I/O classes

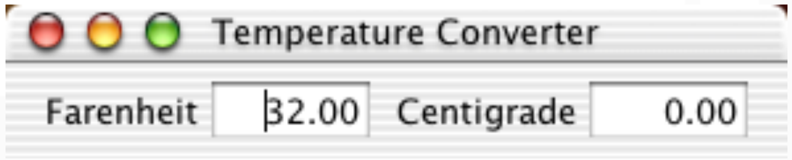
2110 Supplemental: Decorator Pattern

- Examples: Java I/O
  - new PrinterWriter(System.out)
  - new ObjectInputStream(file)



2110 Supplemental: Event-Driven Applications

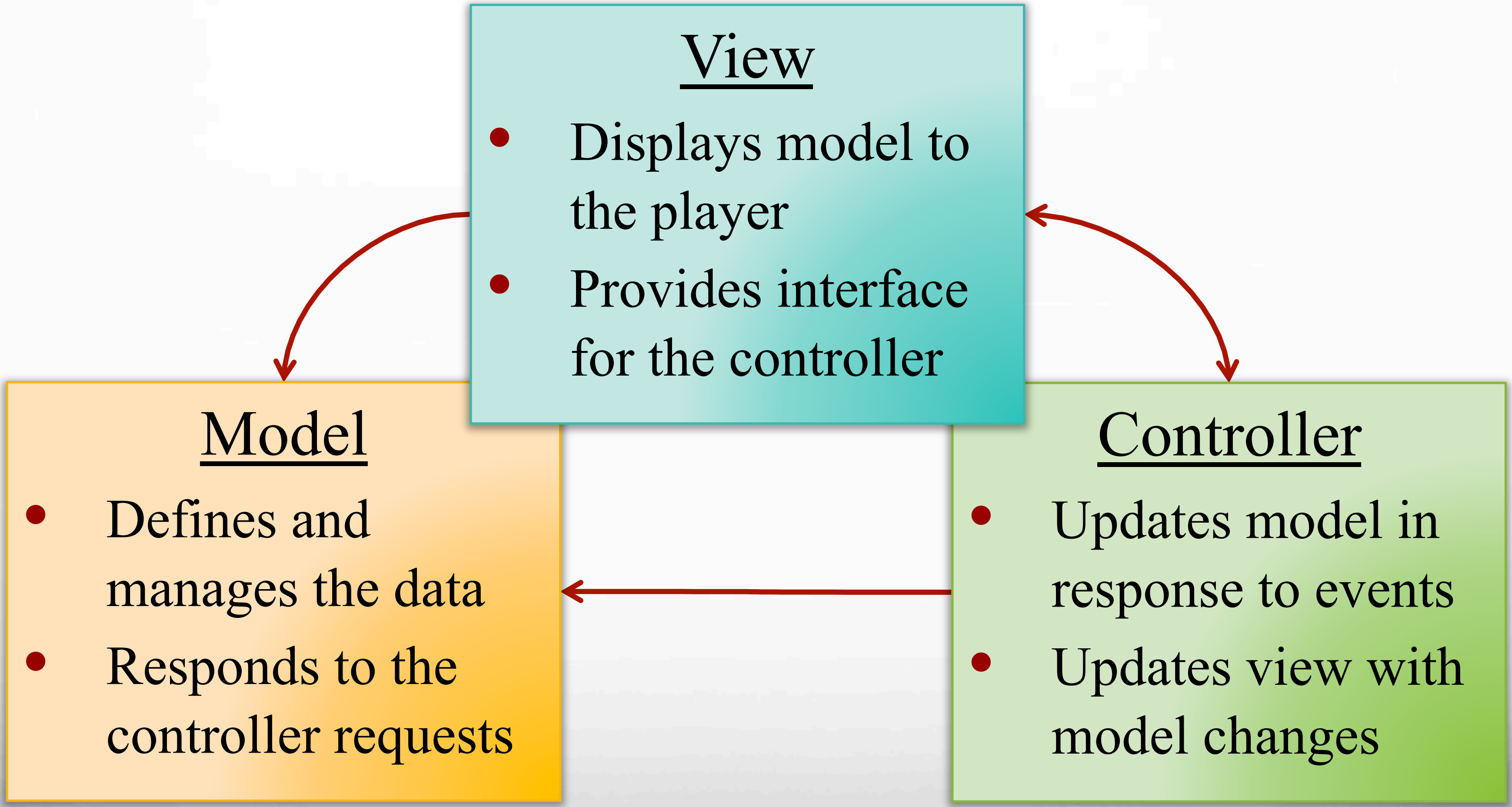
- Most graphical apps are **event driven**



- Each GUI widget can generate **events**
  - Button: Click event
  - Mouse: Click event, move event
- You write **listeners** to react to an event
  - Also called call-back functions
- The OS/VM handles event **detection** for you

2110 Supplemental:

Model-View-Controller Pattern



View

- Displays model to the player
- Provides interface for the controller

Model

- Defines and manages the data
- Responds to the controller requests

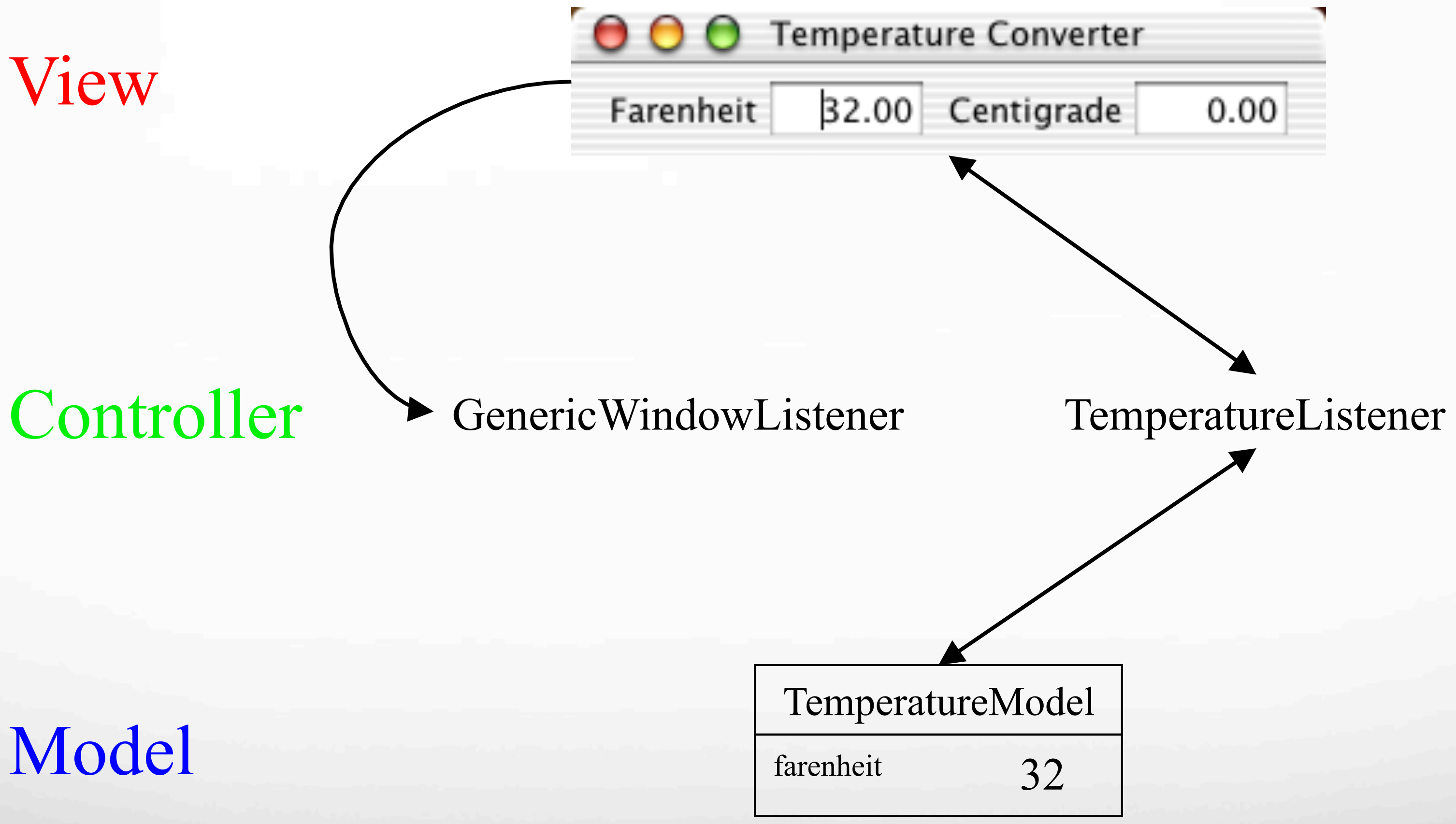
Controller

- Updates model in response to events
- Updates view with model changes

2110 Supplemental: Temperature Converter

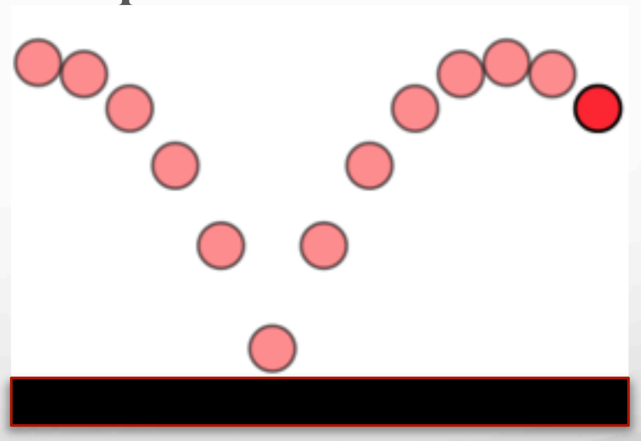
- Model: (TemperatureModel.java)
  - Stores one value: fahrenheit.
  - ADT abstraction presents two values.
- View: (TemperatureConverter.java)
  - Constructor creates objects and connects them.
  - Main method just calls constructor.
- Controller: Two Listeners
  - Respond to window events (GenericWindowListener.java)
  - Keep fields consistent (TemperatureListener.java)

2110 Supplemental: MVC Illustrated



Limitations of Event Model

- Program only reacts to user input
  - Nothing changes if user does nothing
  - Desired behavior for productivity apps
- Games continue without input
  - Character animation
  - Clock timers
  - Enemy AI
  - Physics Simulations

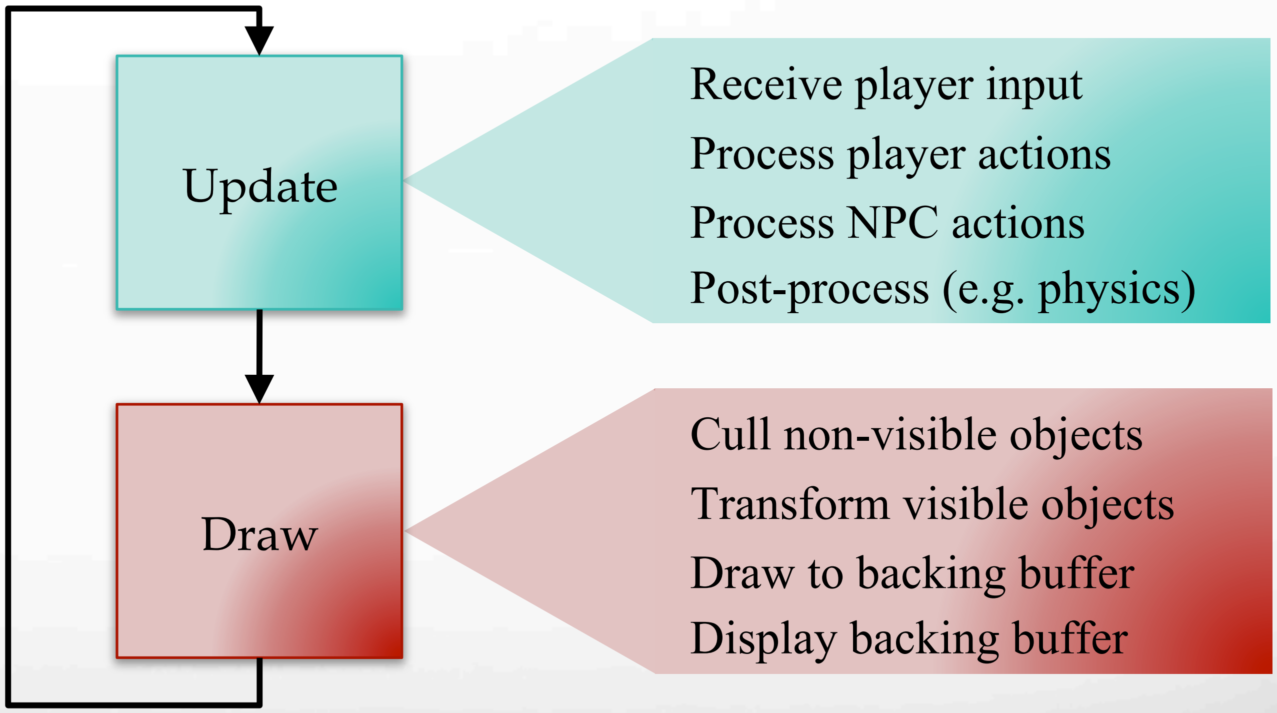


The Game Loop

60 times/s

=

16.7 ms

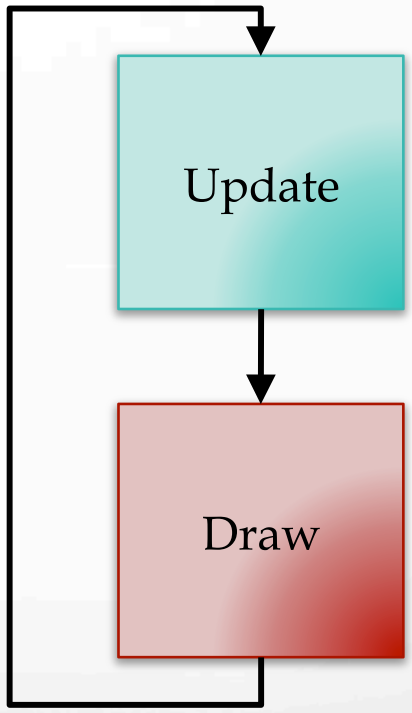


Receive player input
Process player actions
Process NPC actions
Post-process (e.g. physics)

Cull non-visible objects
Transform visible objects
Draw to backing buffer
Display backing buffer

The Game Loop and MVC

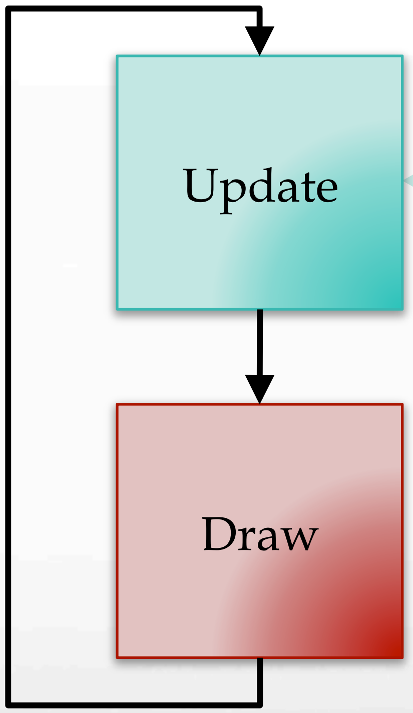
- Model: The game state
  - Value/location of resources
  - What is in the save file
- View: The draw loop
  - Focus of upcoming lectures
- Controller: The update loop
  - Alters the game state
  - Primary topic of this lecture



Few Words on Drawing

- Drawing needs to be fast!
  - Do as little computation as possible
  - But draw as few objects as possible
- Is this a contradiction?
  - Need to compute who to draw
  - So drawing less has extra overhead
- Rule: do **not** modify game state in draw
  - Any extra computation is local-only

The Game Loop



Receive player input

Process player actions

Process NPC actions

Post-process (e.g. physics)

Player Input

- Traditional input is event-driven
  - Events capture state of controller
  - OS/VM generates events for you
  - Listeners react to events

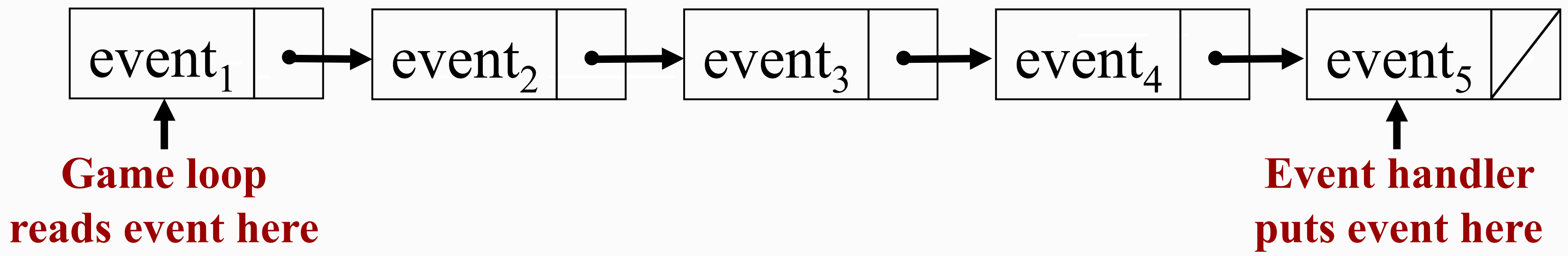
- Game loop uses **polling** for input
  - Ask for controller state at start of loop
    - Example: What is joystick position?
  - If no change, do no actions that loop

Problem with Polling

- Only one event per update loop
  - Multiple events are lost
  - **Example**: Fast typing
- Captures state at beginning
  - Short events are lost
  - **Example**: Fast clicks
- Event-driven does not have these problems
  - Captures all events as they happen

Combining Input Approaches

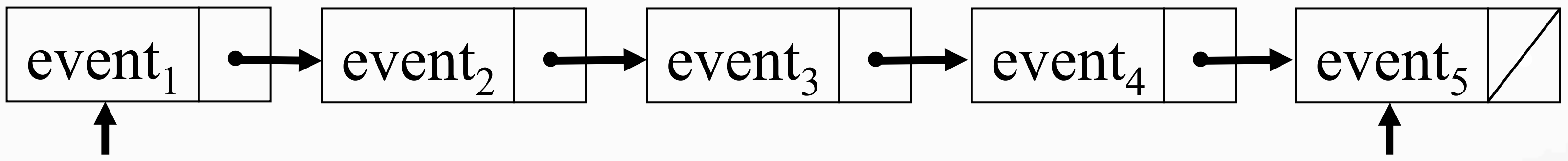
- Can combine using an event queue
  - Listeners write at end of the queue
  - Game loop reads from the front



- Generally requires multiple threads
  - Event handler is (usually) OS/VM provided thread
  - Game loop is an additional thread

Warning: Thread Coordination

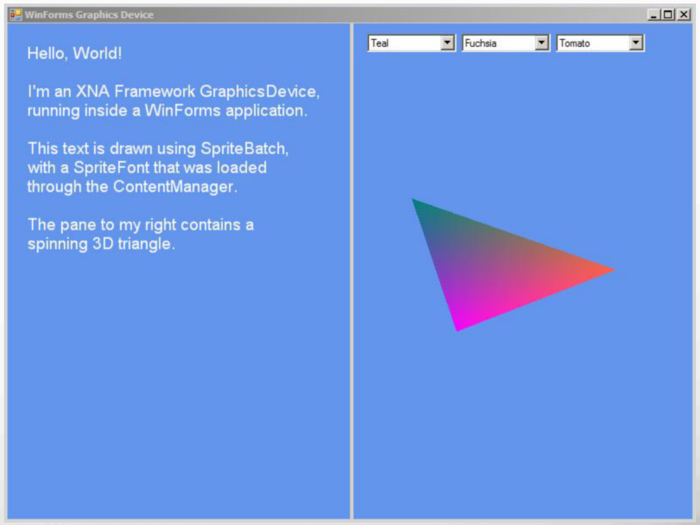
- Threads are tricky if you do not know how
  - Queue is shared between two threads
  - Most queues are not thread safe!
  - What if threads modify queue at same time?



- Classic *critical section* problem
  - Threads need to lock queue when access
  - But locking can be expensive

Warning: XNA Event Handling

- XNA and Windows Forms are different
  - XNA: game loop thread, no event handlers
  - Forms: event handlers, no game loop thread
- Combining is a lot of work
  - Many low-level details
  - Do it only if necessary
- Ruins X-Box compatibility



Hello, World!

I'm an XNA Framework GraphicsDevice, running inside a WinForms application.

This text is drawn using SpriteBatch, with a SpriteFont that was loaded through the ContentManager.

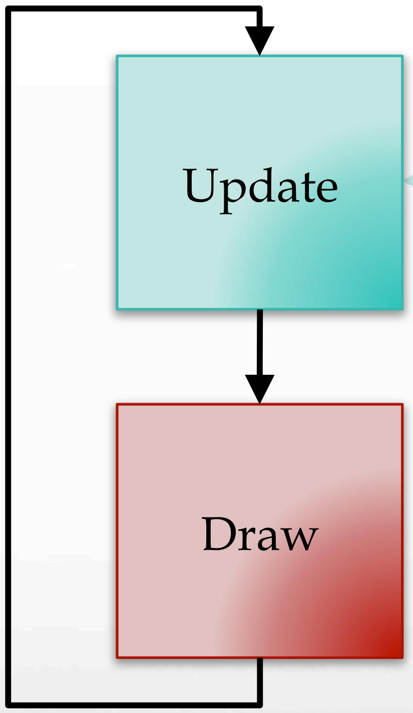
The pane to my right contains a spinning 3D triangle.

Handlers: Really Necessary?

- Most of the time: **No**
  - Frame rate is short: 16.7 ms
  - Most events are > 16.7 ms
  - Event loss not catastrophic

- Buffering is sometimes undesirable
  - Remembers every action ever done
  - But may take a longer time to process
  - If takes too long, just want to abort

The Game Loop



Receive player input

Process player actions

Process NPC actions

Post-process (e.g. physics)

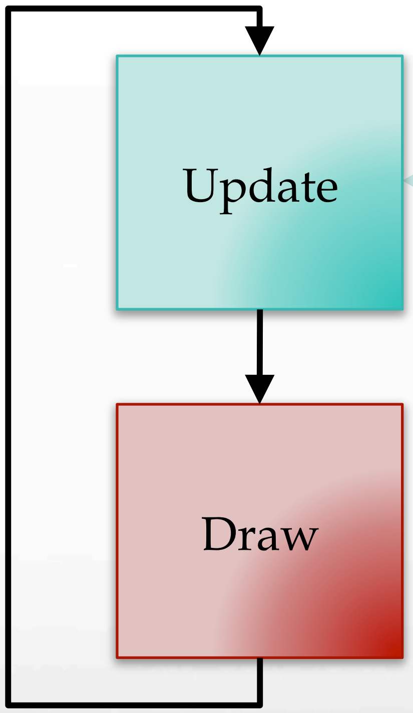
Player Actions

- Actions alter the game state
  - Can alter player state: movement
  - Can alter opponent state: damage
- Player actions correspond to user input
  - Choice is determined by input controller
  - Else action is performed by computer
- These are your game **verbs**!

Abstract Actions from Input

- Actions: functions that modify game state
  - `move(dx,dy)` modifies `x`, `y` by `dx`, `dy`
  - `attack(o)` attacks opponent `o`
- Input controller maps input to actions
  - Read input state from controller
  - Pick an action and call that function
- Input handler should not alter state directly!

The Game Loop



Receive player input

Process player actions

Process NPC actions

Post-process (e.g. physics)

NPC: Non-Player Character

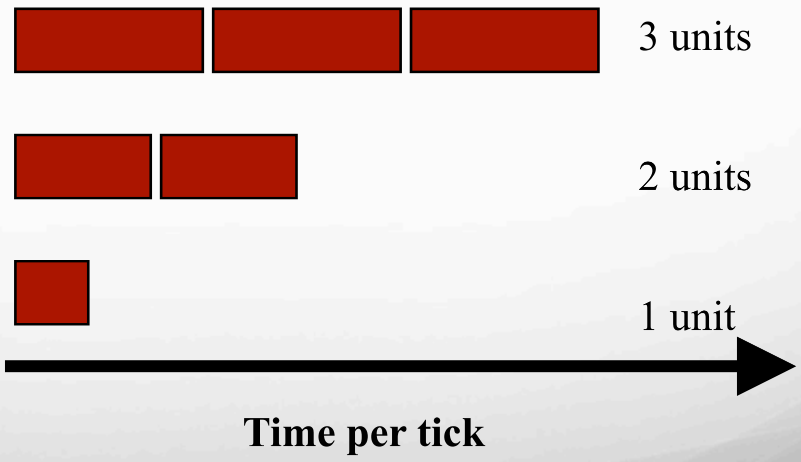
- NPC is an intelligent computer-controlled entity
  - Not a just a physics object
  - Sometimes called an *agent*
- NPCs have their own actions/verbs
  - But no input controller to choose
- Work on sense-think-act cycle
  - **Sense**: perceive the world around it
  - **Think**: choose an action to perform
  - **Act**: update the game state

Sense-Think-Act

- Act should be *very* fast!
  - Simple arithmetic on fields
  - If game is slow, act only
  - **Example**: apply velocity
- Sense-Think more complex
  - Often one unit
  - May be very slow
  - Focus of AI lectures

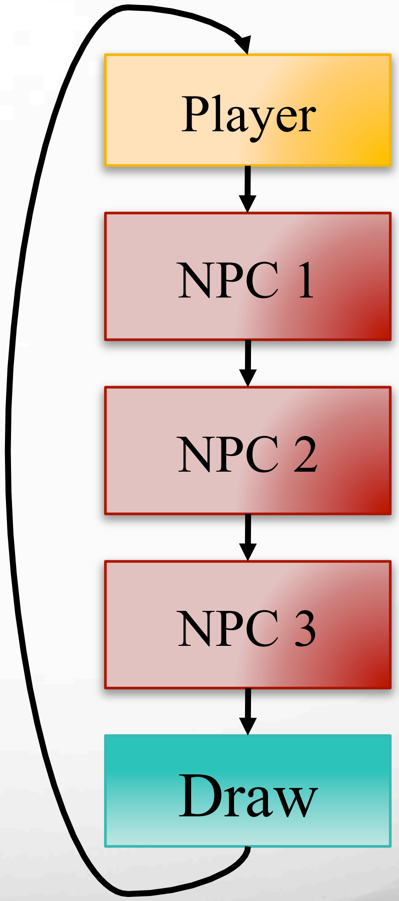
Sense-Think-Act

- Sensing may be slow!
- Example: morale
  - *n* knights, *n* skeletons
  - Knights fear skeletons
  - Proportional to # seen
- Count skeletons in view
  - O(*n*) to count skeletons
  - O(n^2) for all units



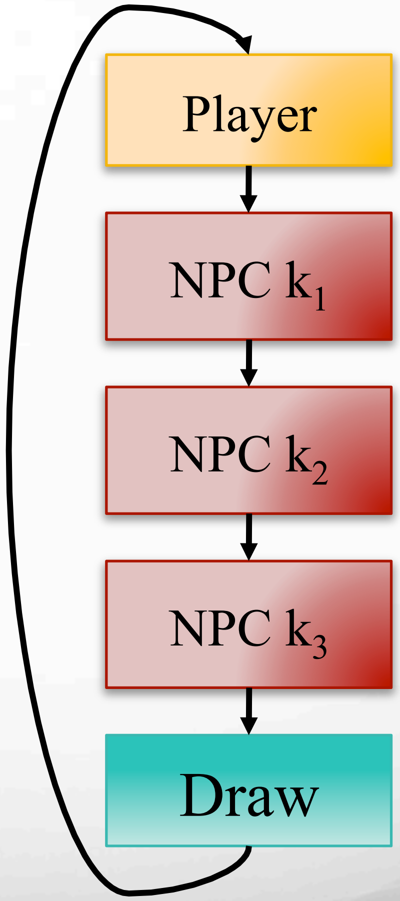
Processing NPCs

- Naïve solution: sequentially
- **Problem**: NPCs react too fast!
  - Each reads the actions of previous
  - Even before drawn on screen!
- **Idea**: only react to what can see
  - Choose actions, but don't perform
  - Once all chosen, then perform



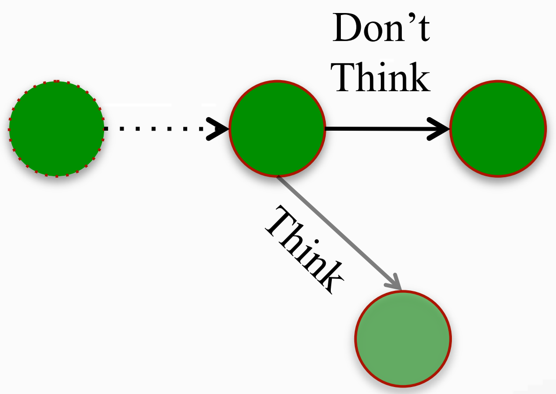
Processing NPCs

- Another idea: long actions
  - Action takes several loops
  - Emulates a thinking delay
- Long actions: naïve solution
  - NPC only acts every *k* loops
- **Problem**: jerky animation
  - Act, don't think



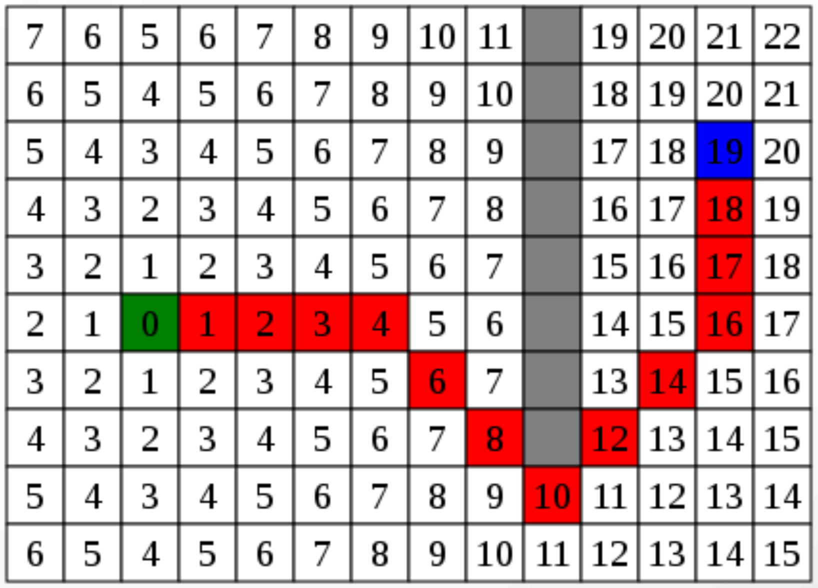
Acting Without Thinking

- Remember last action
  - Keep doing it!
  - Need verb **and** parameters
- Example: Movement
  - Keep track of velocity
  - Apply each game loop
- Dead reckoning



Problem: Pathfinding

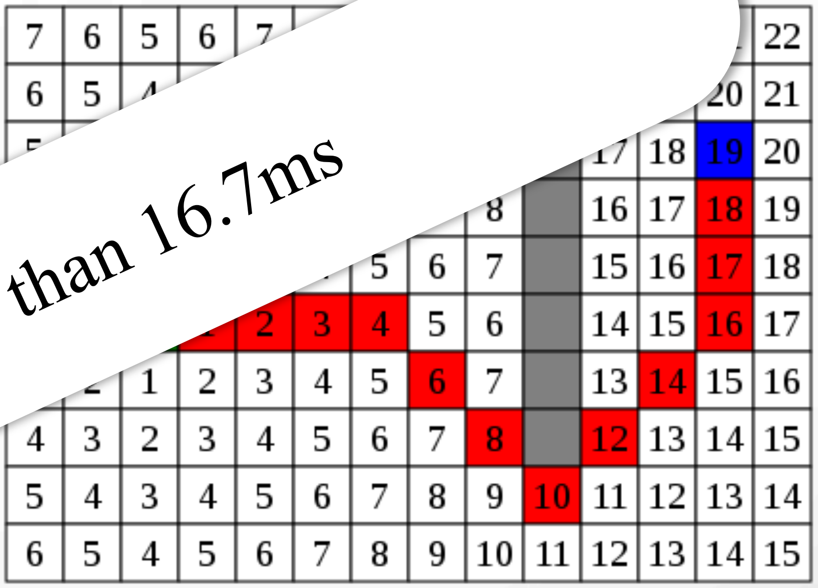
- Focus of Game Lab 2
  - Crucial if top view
  - Major area of research
- Potentially very slow
  - *n* NPCs, *g* grid squares
  - Dijkstra: O(g^2)
  - For each NPC: O(ng^2)



7	6	5	6	7	8	9	10	11		19	20	21	22
6	5	4	5	6	7	8	9	10		18	19	20	21
5	4	3	4	5	6	7	8	9		17	18	19	20
4	3	2	3	4	5	6	7	8		16	17	18	19
3	2	1	2	3	4	5	6	7		15	16	17	18
2	1	0	1	2	3	4	5	6		14	15	16	17
3	2	1	2	3	4	5	6	7		13	14	15	16
4	3	2	3	4	5	6	7	8		12	13	14	15
5	4	3	4	5	6	7	8	9	10	11	12	13	14
6	5	4	5	6	7	8	9	10	11	12	13	14	15

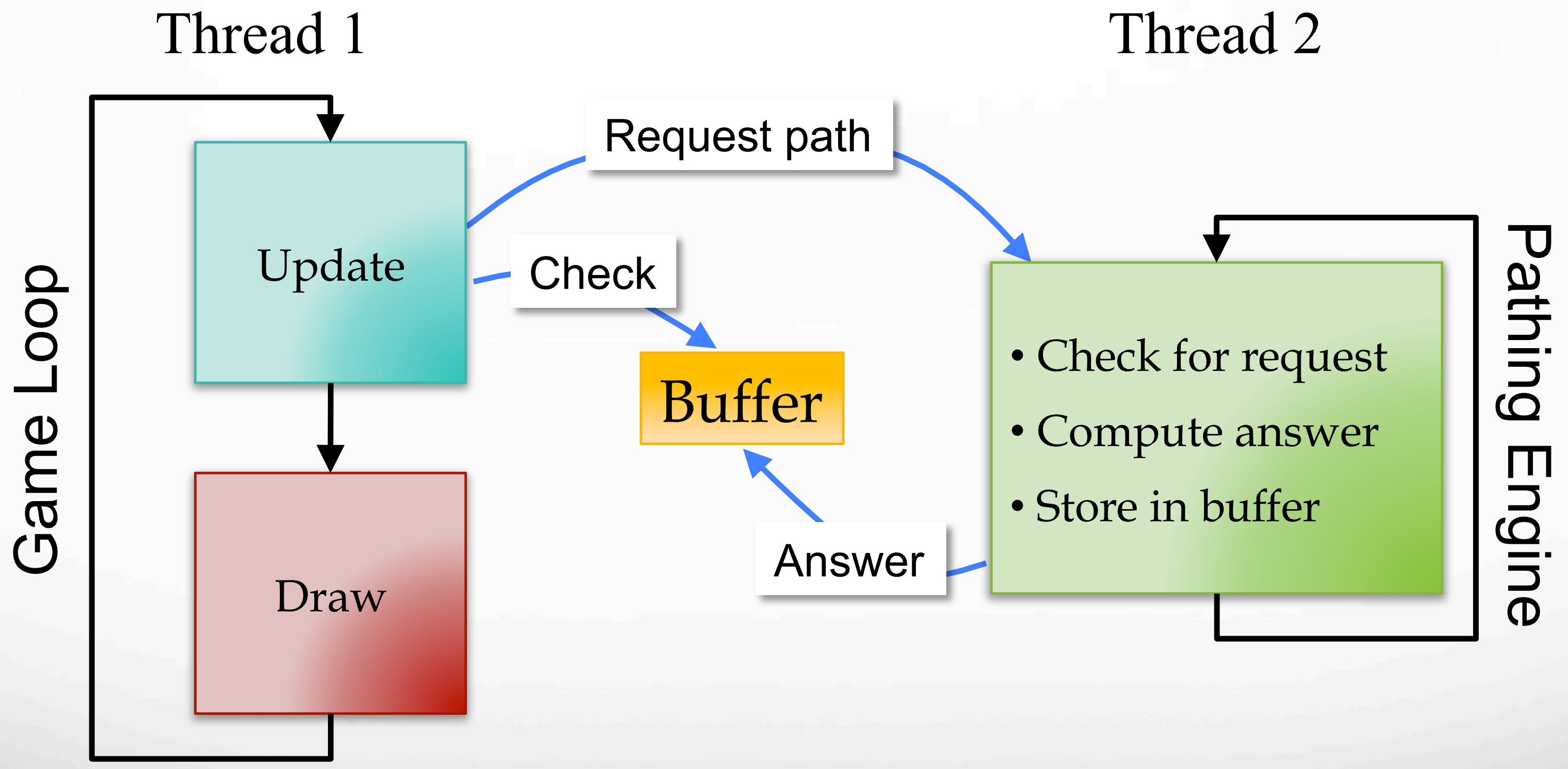
Problem: Pathfinding

- Focus of Game Lab 2
  - Crucial if top view
  - Major area of research
- Potentially very
  - *n* NPCs
  - D
  - F PC: O(ng^2)



Often more than 16.7ms

Asynchronous Pathfinding

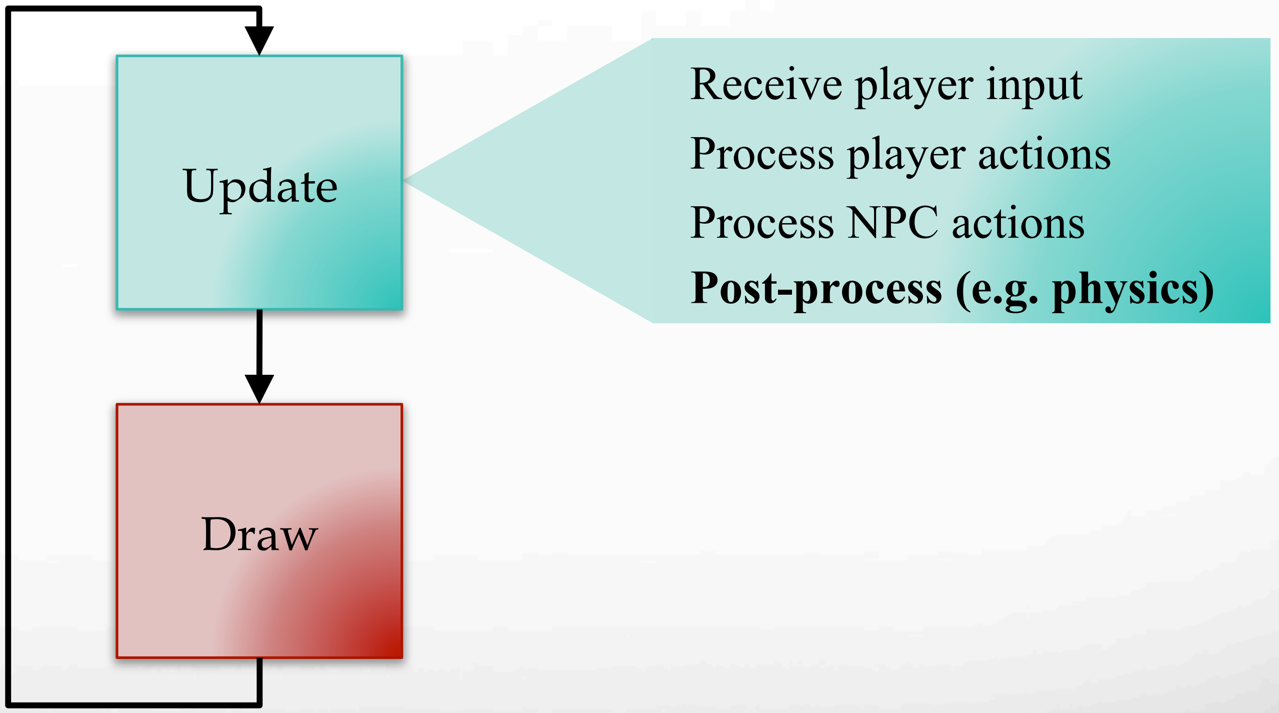


Looks like input buffering!

Asynchronous Pathfinding

- NPCs do not get answer right away
  - Check every loop until answered
  - Remember request; do not ask again
- What to do until then?
  - Act, don't think!
  - If nothing, **fake** something
  - RTS: "Stomping Feet"

The Game Loop



Receive player input

Process player actions

Process NPC actions

Post-process (e.g. physics)

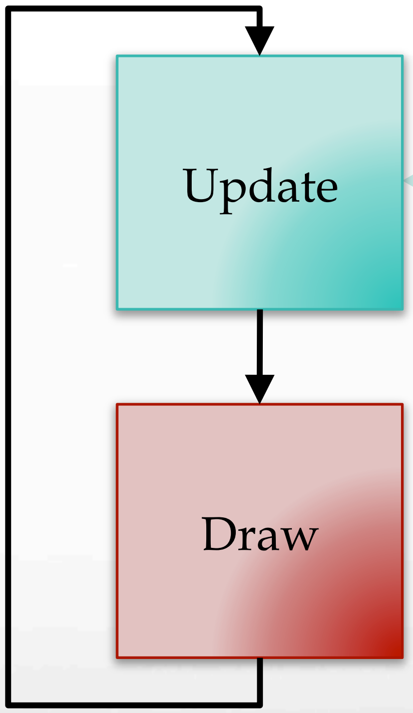
Purpose of a Physics Engine

- Moving objects about the screen
  - Kinematics: Without regard to external forces
  - Dynamics: The effect of forces on the screen
- Collisions between objects
  - Collision detection: Did a collision occur?
  - Collision resolution: What do we do?
- More on this issue later

Physics Engines: Two Levels

- White Box: Engine corrects movement errors
  - Update object state ignoring physics
  - Physics engine nudges object until okay
- Black Box: Engine handles everything
  - Do not move objects or update state
  - Give forces, mass, velocities, etc. to engine
  - Engine updates to state that is close enough

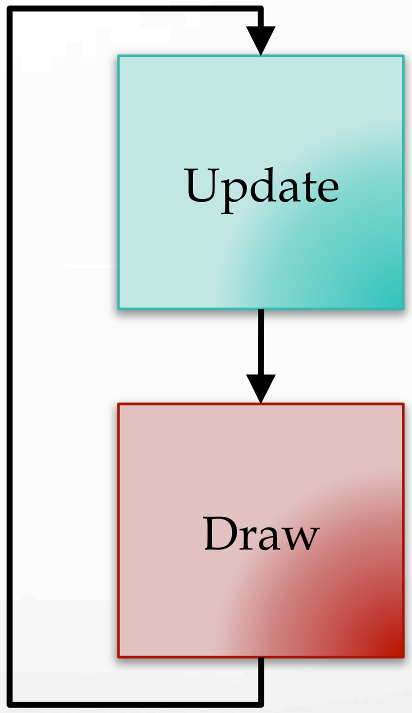
The Game Loop



Receive player input
Process player actions
Process NPC actions
Post-process (e.g. physics)

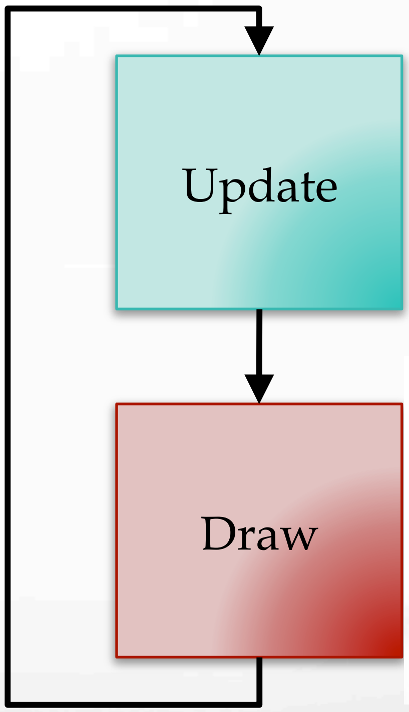
The Game Loop and MVC

- Model: The game state
  - Value of game resources
  - Location of game objects
- View: The draw loop
  - Focus of upcoming lectures
- Controller: The update loop
  - Alters the game state
  - Primary topic of this lecture

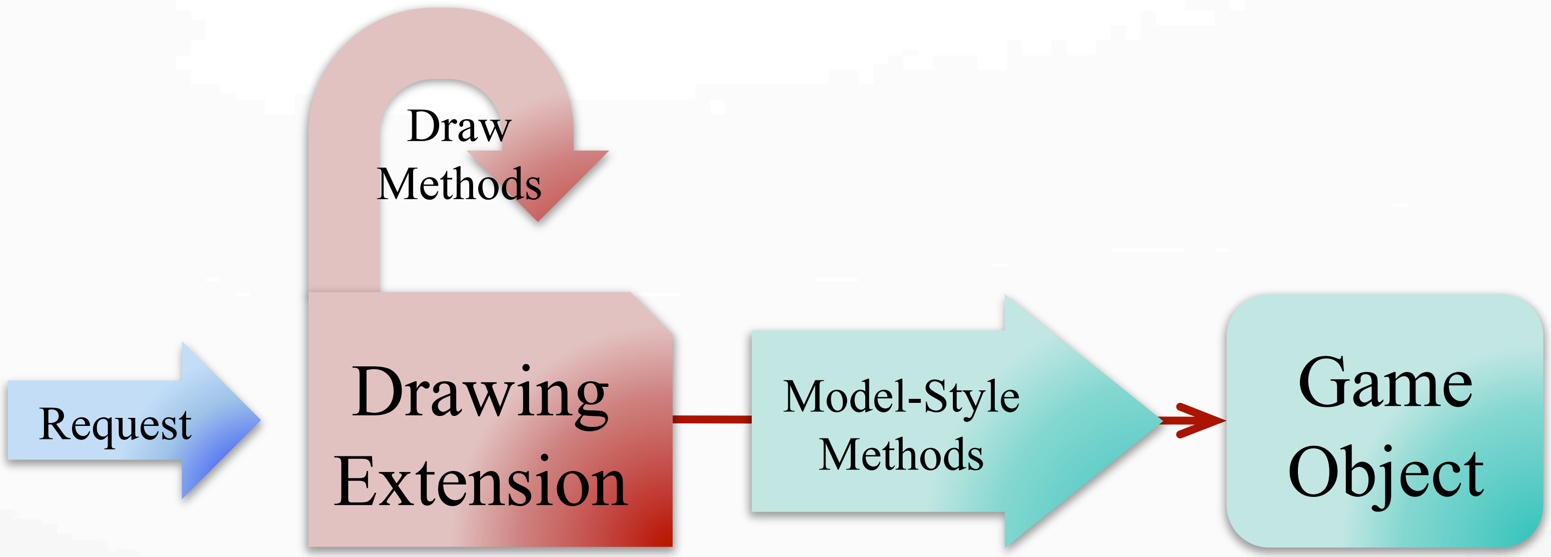


Not So Fast

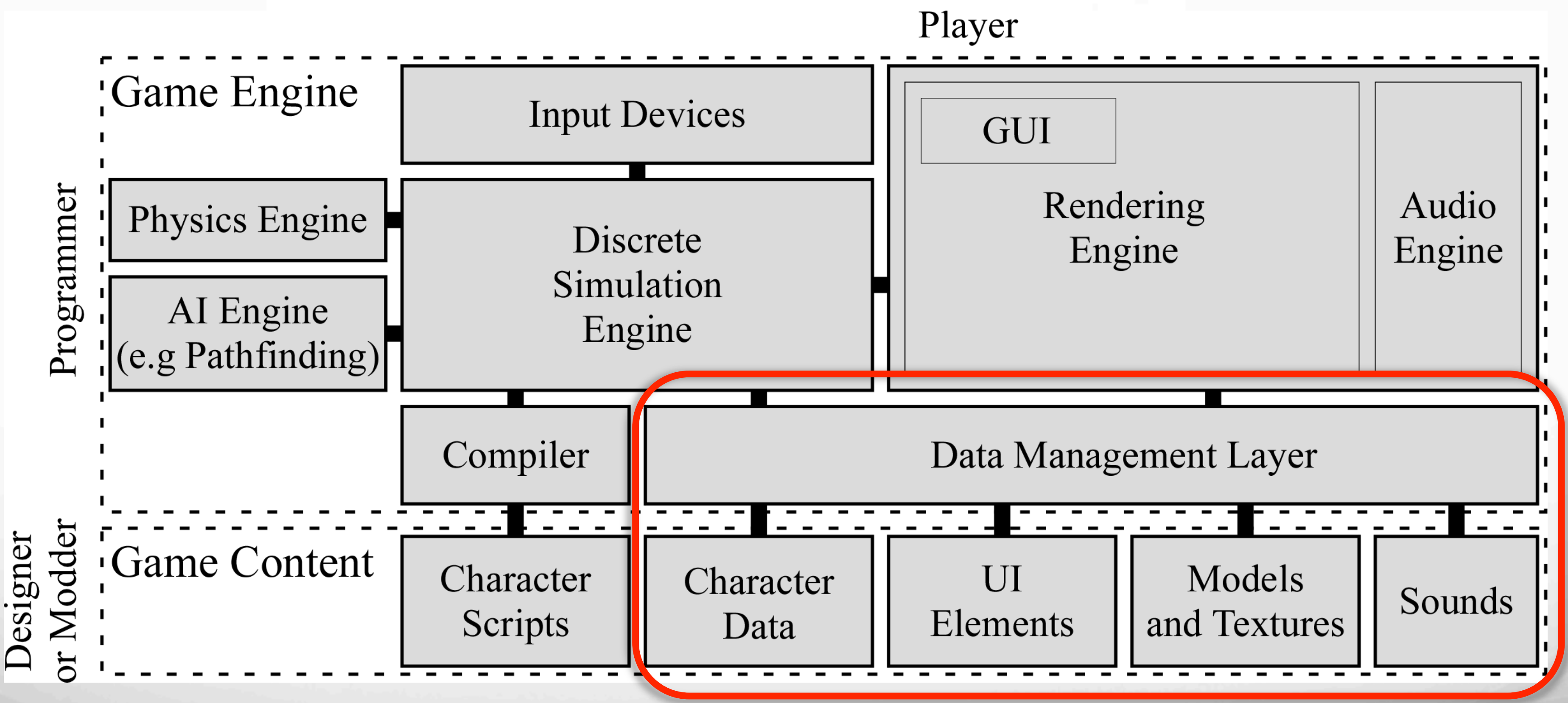
- Way too much to draw
  - Backgrounds
  - UI elements
  - Individual NPCs
  - Other moveable objects
- Cannot cram all in Draw
  - Put it in game object
  - But objects are models



Decorator Pattern Revisted



Is that Everything?



Data Management

- A lot of it handled for you automatically
  - XNA supports standard graphics formats
  - XACT format used for sounds
- Except the data that you create!
  - Save files (for your player)
  - Game levels (for the level editor)
- Make all models/game state **serializable**

Future Lectures

- We will spend the semester filling in details
  - Data-Driven Design: Data Management
  - 2D Graphics: Drawing
  - Character AI: Sense-Think-Act cycle
  - Strategic AI: Asynchronous AI
  - Physics Engines: Collisions, Forces
  - Networking (at end of course)
- But there is more design coming too