CISC 1600 Lecture 2.4
Introduction to JavaScript

Topics:

Javascript overview

The DOM

Variables and objects

Selection and Repetition

Functions

A simple animation

What is JavaScript?

- JavaScript is not Java
- JavaScript was designed to add interactivity to HTML pages
  - It has grown beyond that to run outside the browser
- JavaScript is usually interpreted
  - but Google created an open source compiler for it (V8)
- JavaScript is a web standard

What can JavaScript do?

- Insert dynamic text into an HTML page
- Read and write HTML elements
- React to events: mouse, keyboard, network, etc.
- Asynchronously load or update parts of pages
- Validate form data before it is submitted
- Examples for class here

Popup boxes

JavaScript allows interactivity through the use of pop-up boxes

There are 3 general kinds:

```
alert("sometext");
confirm("sometext");
// returns "true" or "false")
prompt("sometext","defaultvalue");
//returns what user enters as answer
```

See 6_PopUps

JavaScript general syntax

- Syntax is much like Processing (or C or Java)
- Statements “should” end with a semicolon ;
- White space does not matter (except for line breaks sometimes)

```
/*****************************
This is JavaScript (JS), the programming language that powers
the web (and this is a comment, which you can delete).
*****************************/
function greetMe(name) {
  var today = new Date();
  alert("Hello " + name + ", today is " + today.toDateString());
}
```

JavaScript can be defined and used in the body or the head of the page

```
<html>

    <head>
        <script type="text/javascript">
            function message()
            {
                alert("This alert box called by
onload event");
            }
        </script>
    </head>

    <body onload="message()">
        <script type="text/javascript">
            document.write("Message written by
JavaScript");
        </script>
    </body>

</html>

<!-- See 2_EventListeners-->
```

Interactivity: events and listeners

- JavaScript code can respond to “events”
- Events happen in response to
  - User-interface interactions (mouse, keyboard)
  - Network operations (page or resource loading)
- Can register “event listener” functions
  - Called when event happens with information on event

```
<button type="button" onclick="go()">Go!</button>
```

Recall: Document Object Model (DOM)

- A web browser interprets your HTML
  - And builds a model of the page, the DOM
- The DOM is what is rendered to the screen
- The DOM can be manipulated by CSS and JavaScript after it is built
- When building a page, consider its structure first, i.e., the DOM

JavaScript can manipulate the DOM

- Several ways to select DOM elements
  - `document.getElementById("idOfElement");`
  - `document.getElementsByTagName("div");`
  - `parent.children[index];`
  - Etc.
- Several ways to modify DOM elements
  - `element.innerHTML = "Some HTML";`
  - `img.src = "newUrl";`
  - `element.style.border = "1pt solid";`

Simple DOM manipulation

```
<html>
<head>
</head>
<body>
    <script type="text/javascript">
            document.write("<h1>Hello World!
</h1>");
    </script>
</body>
</html>

<!-- Note use of document object model -->
<!-- Example Here:  1_FirstJavaScript.html
-->
```

Variables hold values

- Variables are named locations for storing data
- Assign a value to a variable to refer to it later
  - Values can be simple: number, string, boolean
  - Or complex: DOM node, document object, array of other values
- Create a variable: `var count;`
- Assign a value to it: `count = 0;`
- Use it: `newH1 = "...count = " + count + "</h1>";`
- JavaScript in the browser has several pre-defined variables, e.g.,
  - window: the browser window/frame around the page
  - document: the DOM and other page characteristics

Some JavaScript syntax for math, logic, etc.

- Mathematical:
  `+, -, *, /, %, ++, --`
- Logical:
  `&&, ||, !, <, >, etc...`
- Variable declaration and assignment:

```
var x=10;
var y=2;
var z=x/y;
```

- Strings can be concatenated with the '+' sign.

```
var txt1 = "What a very";
var txt2 = "nice day";
var txt3 = txt1 + " " + txt2;
```

- Note: If you add a number and a string, the result will be a string!

JavaScript is object-oriented

- Like Processing, you can define new data types in JavaScript as objects
- An object is a collection of facts and functions
  - Facts: data, the state of the object
  - Functions: the available operations to perform on it
- In JavaScript, both are accessed using a dot '.'
  - Fact: document.children
  - Function: document.getElementById()
- An object can contain simple values or other objects

Selection: The ability to make a choice

- Execute code only if certain conditions are true
- This is mainly the “if” statement in languages

```
if ((count % 3) == 0) {
    target.innerHTML = target.innerHTML +
newH1;
} else {
    target.innerHTML = newH1;
}
```

Repetition: The ability to repeat an action

- Repeatedly execute code, modifying state
- In JavaScript: mainly “for” and “while” loops

```
var stars = "";
for (var i=0; i < count; i++) {
  stars = stars + "*";
}
// OR
while (stars.length < count) {
  stars = stars + "*";
}
```

Functions: reuse operations

- JavaScript implements the procedural paradigm
  - Because you can write and use procedures (functions)
- Functions take input arguments & return outputs

```
function replaceInnerHtml(nodeId, html) {
  var target = document.getElementById(nodeId);
  target.innerHTML = html;
  return target;
}
```

- Functions allow code to be more modular and reusable
- Debug a function once, use it many times

Arrays are data structures that store multiple values

- Arrays in JavaScript are like arrays in Processing

```
var langs = ['c', 'c++', 'java'];
for (var i=0; i < 3; i++) {
    console.log(langs[i]);
}
```

- Unlike Processing you can put things with different types into the same array in JavaScript

```
var misc = [1, 'a', true];
for (var i=0; i < 3; i++) {
    console.log(misc[i]);
}
```

CISC 1600 Lecture 2.5
Programming languages

Topics:

Multimedia computing tasks
Programming paradigms
Languages we have seen

Some multimedia computing tasks

- Compute a number (simulation)
- Draw a picture, generate a sound (synthesis)
- Create a game (interactivity)

How do you make a computer do that?

- Computers read binary instructions and data
- But computers let us build powerful tools
  - Like programs to make programming easier!
- Compiler: a program that converts programs written in human-readable form into machine-readable form
- Interpreter: a program that runs human-readable programs directly
- Integrated development environment

Programming languages vary by...

- High-level vs low-level
  - High-level: closer to how people think
  - Low-level: closer to how computers operate
- Programming language vs scripting language
  - Programming: compiled into machine code
  - Scripting: interpreted directly
- The programming paradigms they facilitate

Programming paradigms

- Programming paradigm: a style of programming
  - How you define a program's structure & elements
  - Hundreds of different ones
  - Each language can support several / many
- We focus on 4 (but you may not have noticed!)
  - Imperative: a smart list
  - Event-driven: responding to events
  - Procedural: sending messages
  - Object-oriented: a set of interacting objects

Imperative: a smart list

- Tell the computer **how** to perform a task
- Contrast with
  - Declarative: telling it **what** to do, it figures out how
  - Event-driven: responds to interrupting events
- Imperative languages need three things
  - Sequence: an order in which to process information
  - Selection: the ability to make a choice (“if” statements)
  - Repetition: the ability to repeat an action (“for” loops)

Event-driven: responding to events

- Instead of executing a program from top to bottom, execute in response to the environment
- Events can be
  - User input:
  - Network events:
  - Messages from other programs:
  - Sensor events:

Event-driven: responding to events

- Instead of executing a program from top to bottom, execute in response to the environment
- Events can be
  - User input: key pressed, button clicked
  - Network events: request, response, activity
  - Messages from other programs: like network, but on the same computer
  - Sensor events: sound or video frame recorded

Procedural: sending messages

- One section of the program can send a message to another section & receive a well-defined response
- Allows for code reuse, modularity
- Types of procedure calls
  - Function calls within a program
  - Message passing between objects / agents
  - Remote procedure calls (e.g., REST) over networks
- My advice
  - Write lots of short single-purpose procedures
  - Don’t repeat yourself

Object-oriented: interacting objects

- Visualize a program as a set of interacting objects
- Identify **facts** and **functions** of each object
  - Facts: properties of objects, current state
  - Functions: procedures, possible operations
- Objects are instances of classes
- ChessPiece is a class, each piece on board is an object
  - Facts: current position on board
  - Functions: possible next moves

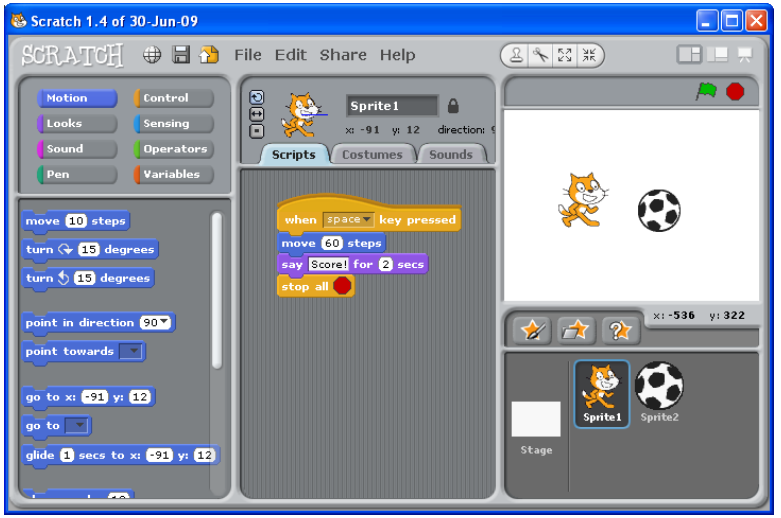
These paradigms are of different types

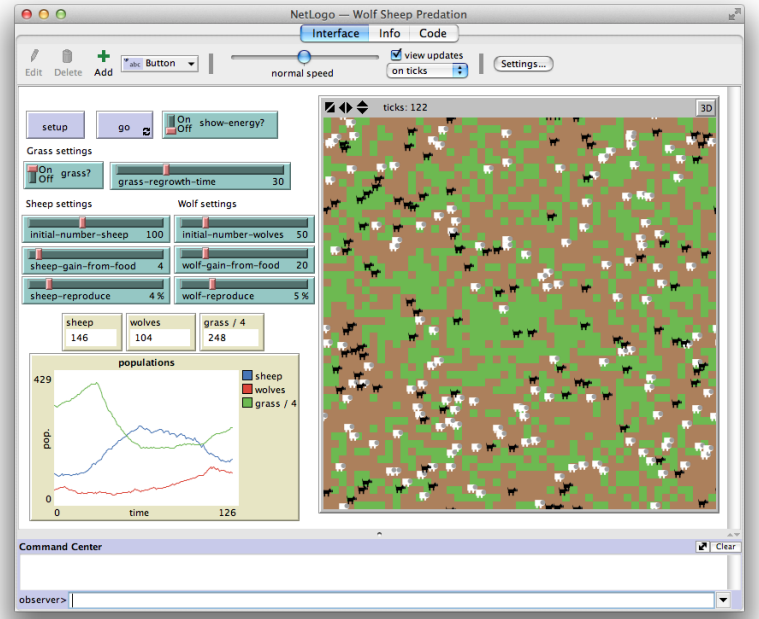
- Control flow: interactivity
  - Imperative
  - Event-driven
- Code organization: modularity and reuse
  - Procedural
  - Object-oriented

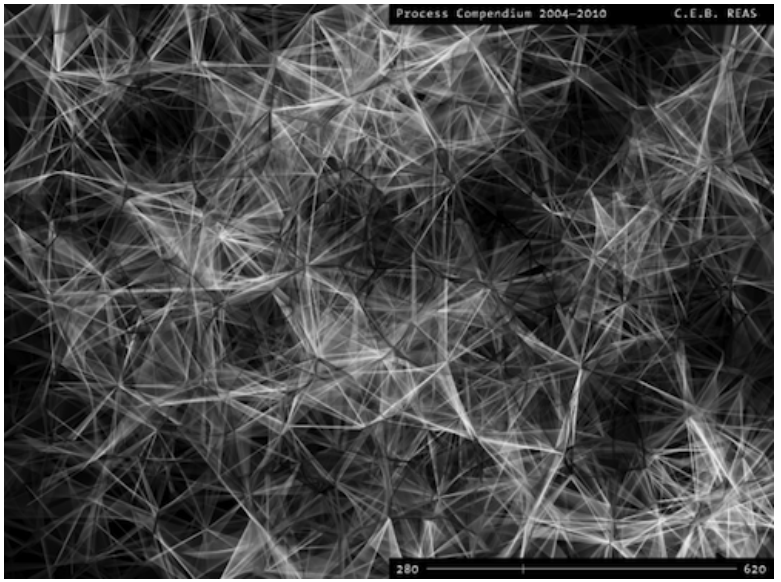
Which paradigms fit these tasks?

- Compute a number (simulation)
- Draw a picture, generate a sound (synthesis)
- Create a game (interactivity)

Programming languages in this course



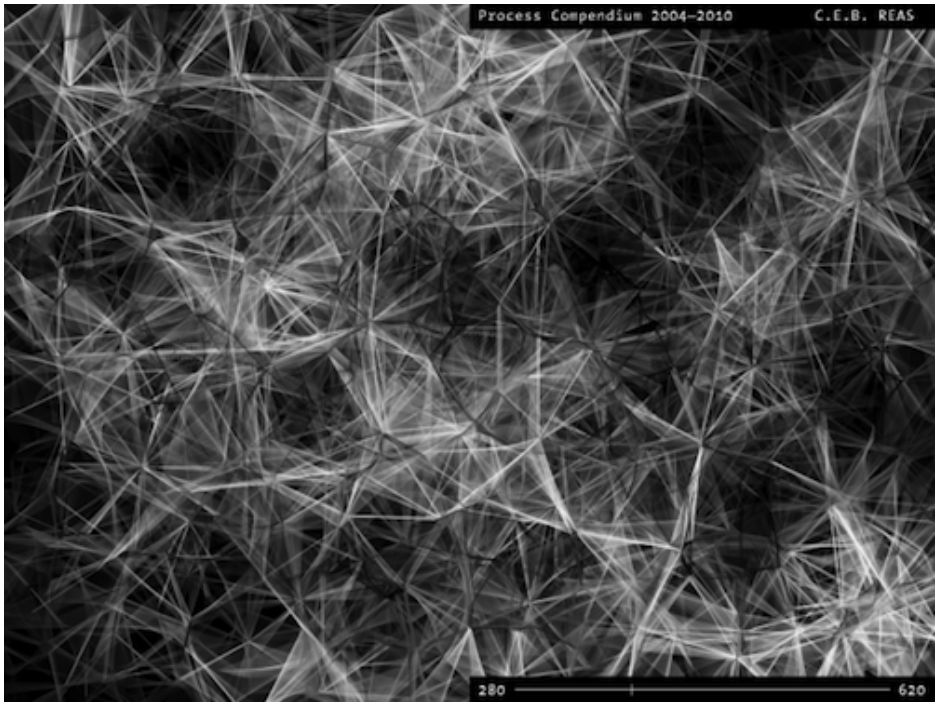




JavaScript

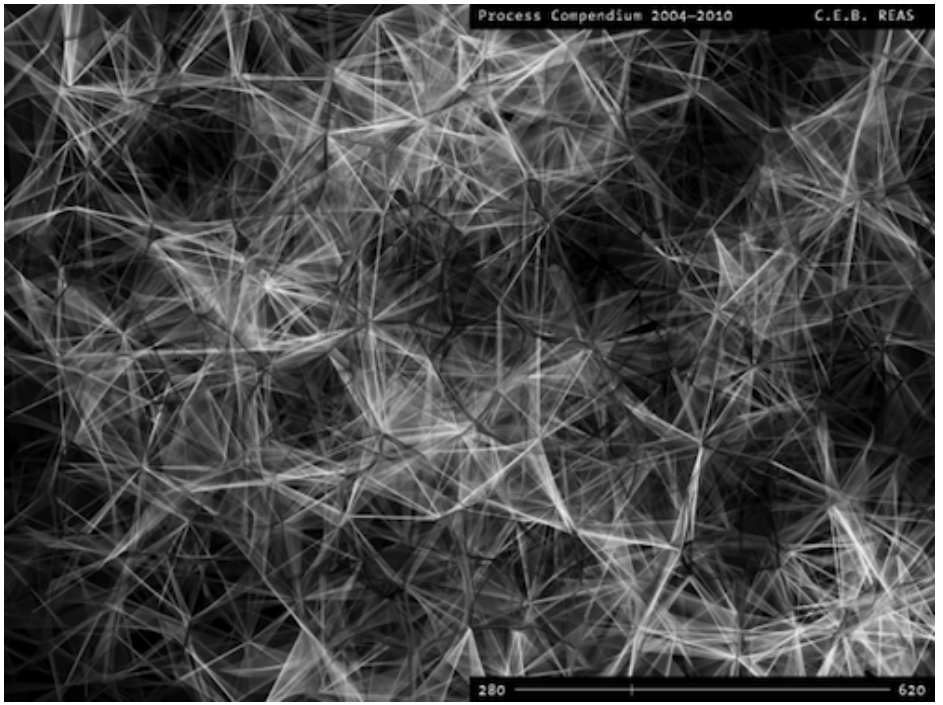
Processing

- Goal: draw pictures, animations
- Interactivity: moderate
  - can be event driven
- Modularity: high
  - object oriented



Processing

- Goal: draw pictures, animations
- Interactivity:
  -
- Modularity:
  -



Javascript

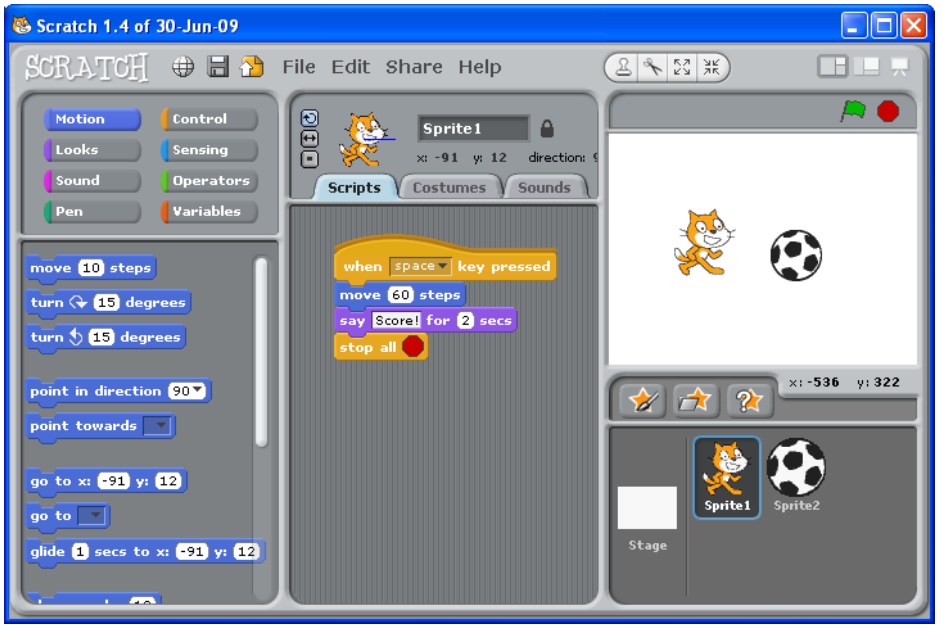
- Original goal: add interactivity to web pages
- New goal: general purpose, cross-platform computing
- Interactivity: high
  - event-driven, imperative
- Modularity: high
  - object oriented, procedural

Javascript

- Original goal: add interactivity to web pages
- New goal: general purpose, cross-platform computing
- Interactivity:
  -
- Modularity:
  -

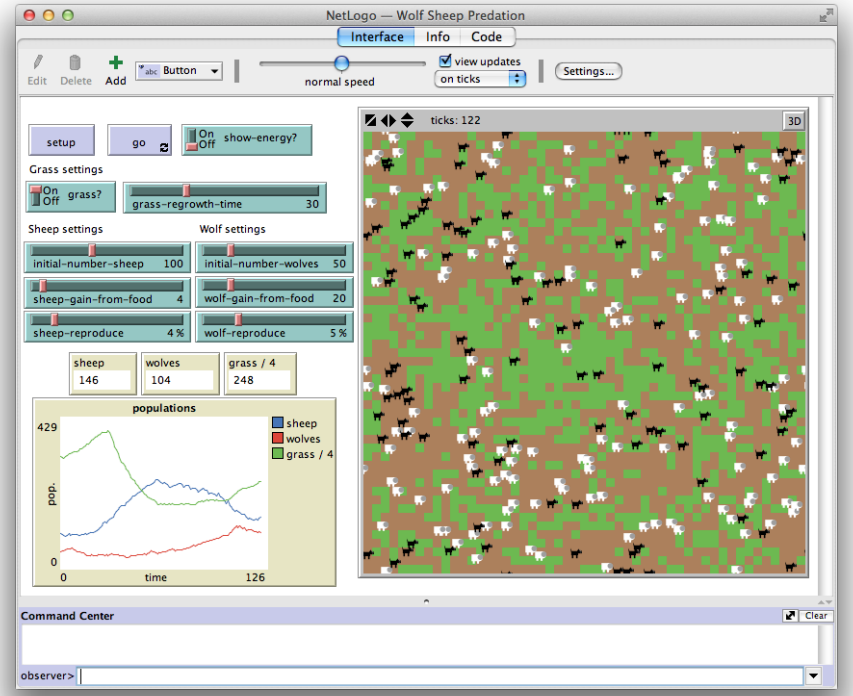
Scratch

- Goal: build interactive games, animations
- Interactivity high: event-driven, imperative
- Modularity high: sprites are objects, agents



NetLogo

- Goal: simulate interacting “agents”
- Interactivity low: imperative
  - Agents interact with each other
- Modularity high: object oriented, agent-based



Summary

- Different programming tasks are easier in different programming paradigms
- Different programming paradigms are easier in different programming languages
- We will cover a decent variety of languages in this course that facilitate the paradigms of
  - Imperative, event-driven, procedural, and object-oriented
- (Don’t repeat yourself)

CISC 1600 Lecture 3.1
Introduction to Scratch

Topics:

Scratch interface

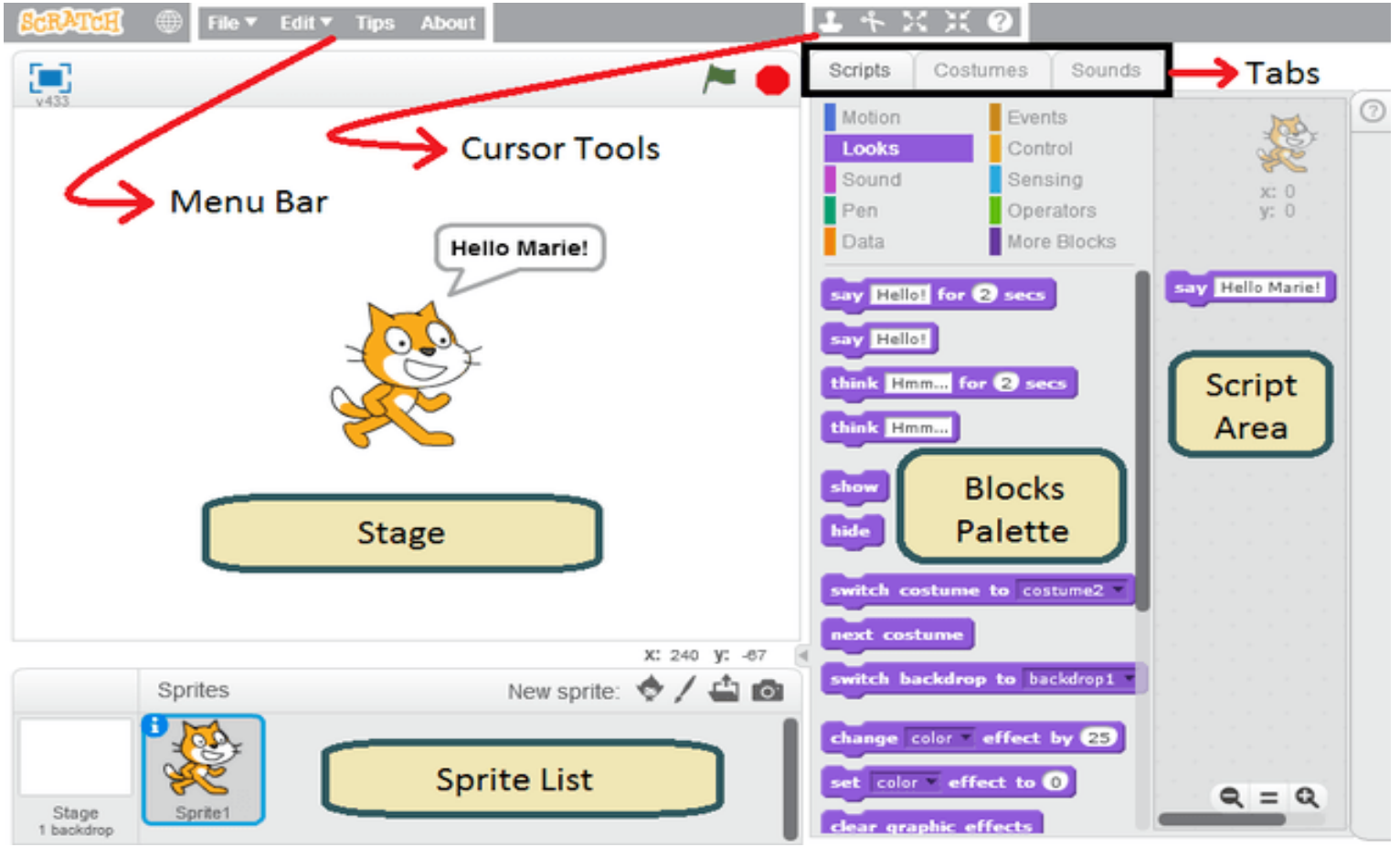
Programming in Scratch

Examples Scratch projects

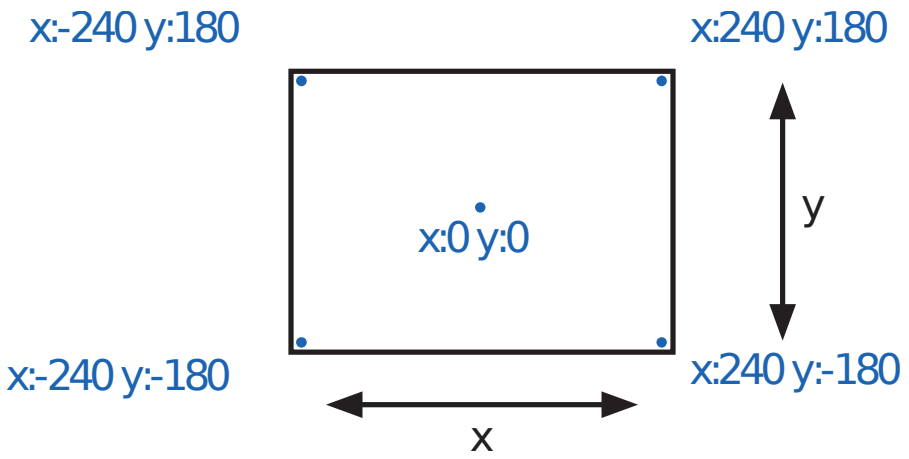
Scratch

- Scratch is visual programming language for creating games and animations
- It has a powerful integrated development environment that will help you work faster
- It makes it easier to do many things we were doing "manually" in Processing
  - Collision detection, movement, bouncing, animation
- It supports many of the same ways of thinking about programs as Processing and JavaScript

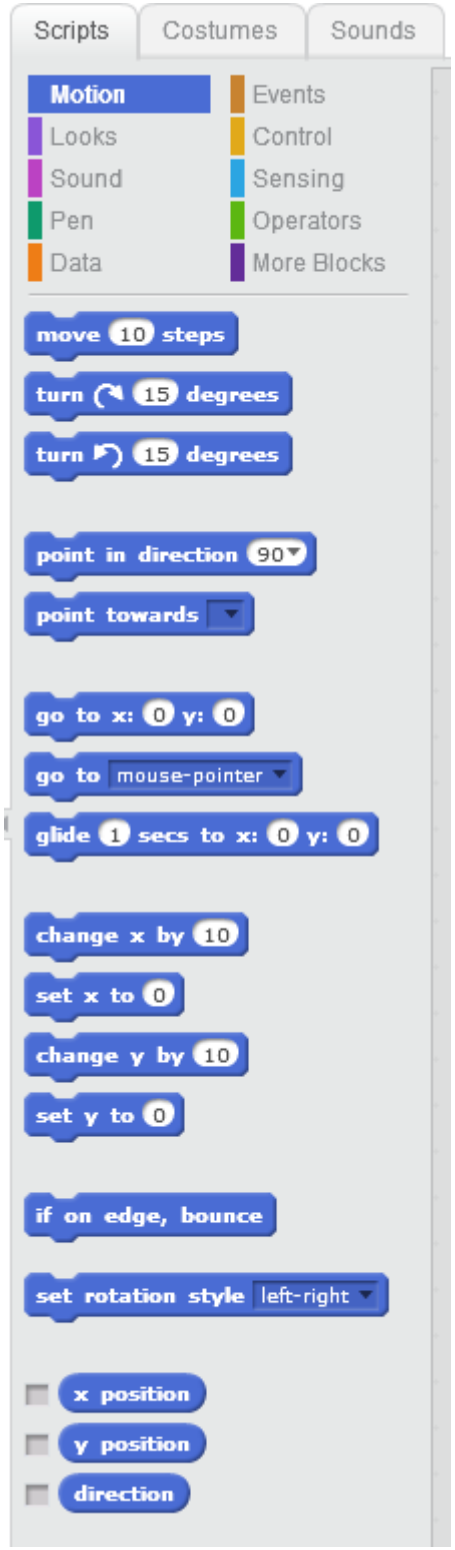
Scratch Interface



The stage



- Like Processing, Scratch has a Stage (Canvas)
- The Stage is 480 units wide and 360 units tall.
- Unlike Processing, (0,0) is the middle of the stage

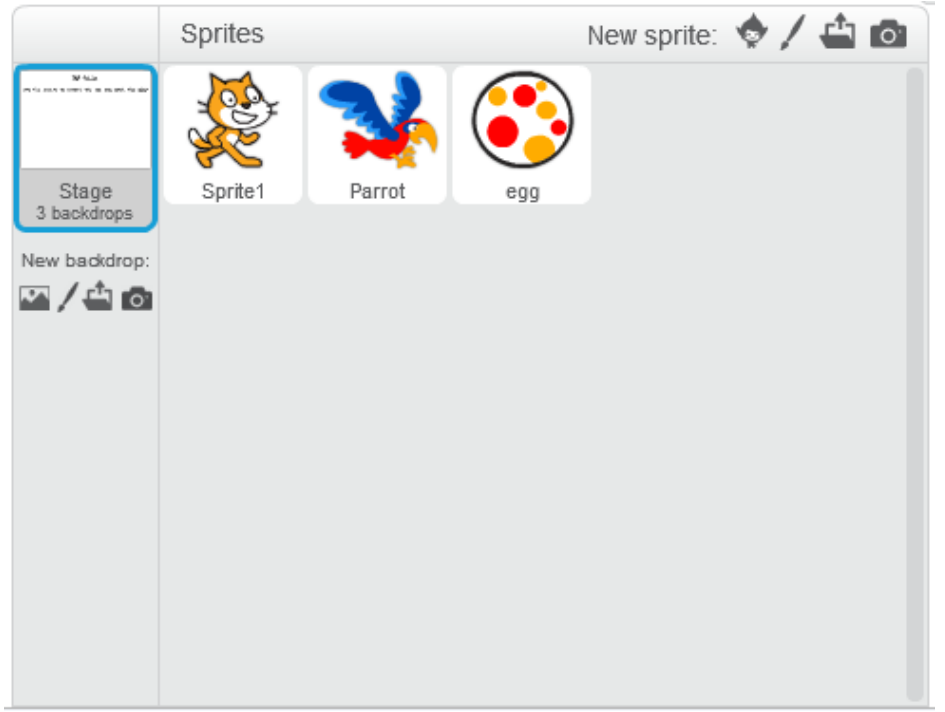


Code blocks and scripts

- Code is written by assembling Blocks into stacks called Scripts
- Stacks are executed from the top
- To run a block or a stack, click on it
- Multiple scripts can run at the same time. Running scripts are highlighted
- Some blocks have values that you can edit as text, or put other blocks in

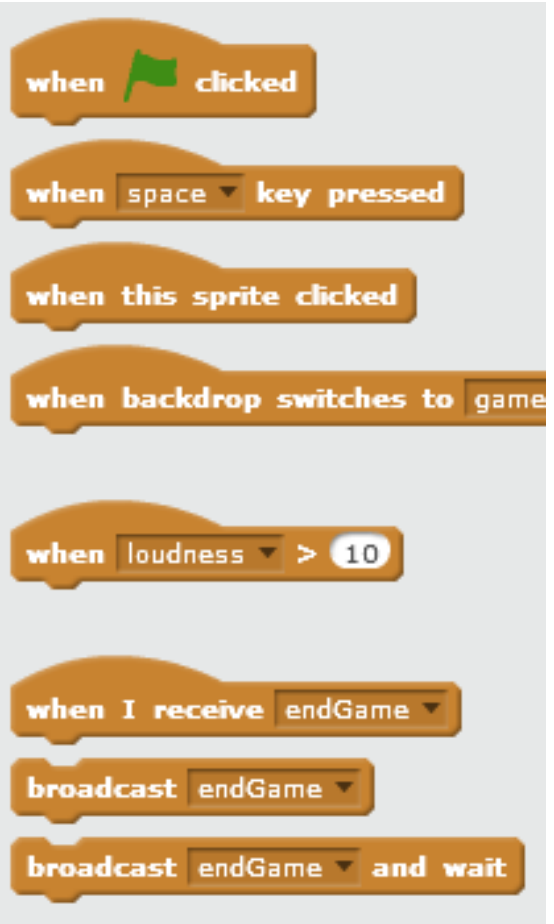
Objects in Scratch: Sprites

- Sprites have Facts and Functions
- Some facts are built-in
  - Position, rotation, pen
- You can add more by adding variables
- Functions are scripts specific to each sprite



Sprites talk to each other in Messages

- Main message is the start flag
- Any sprite can send a message to any other sprite
- Messages start scripts running
- Also use messages in animations to cue events
  - Best to have a single message source in this case



Are you an Artist?
(not necessarily a 'gamer')

- Using Sketch you can develop movies, animations and visual storyboards.
  - What's your story?
  - Who are the characters?
  - What's the conflict?
  - What do you want your audience to feel?
- What objects would you need to create?
- What would their properties/functions be?

Designing a Scratch project

Object	Facts	Functions
• Name • Description	• What are the facts about this object? • What does the object look like? • How many images will you need for it? • Where does it start? • What are its states (alive, dead, etc.)	• What does this object do? • Can it move? • Can it change costumes? • Can it interact with other objects? • Can it interact with the player?
• **Cat** • Controlled by player	• Position • Orientation • 2 costumes for walking • Starts at (0,0)	• Follows mouse • Changes costume to walk • Bounces off edge

CISC 1600 Lecture 3.2
Game design

Topics:

The business of Games

“Funativity”

Concrete components

Narrative

Mechanics, Dynamics, & Aesthetics

•Video Games = Big Business

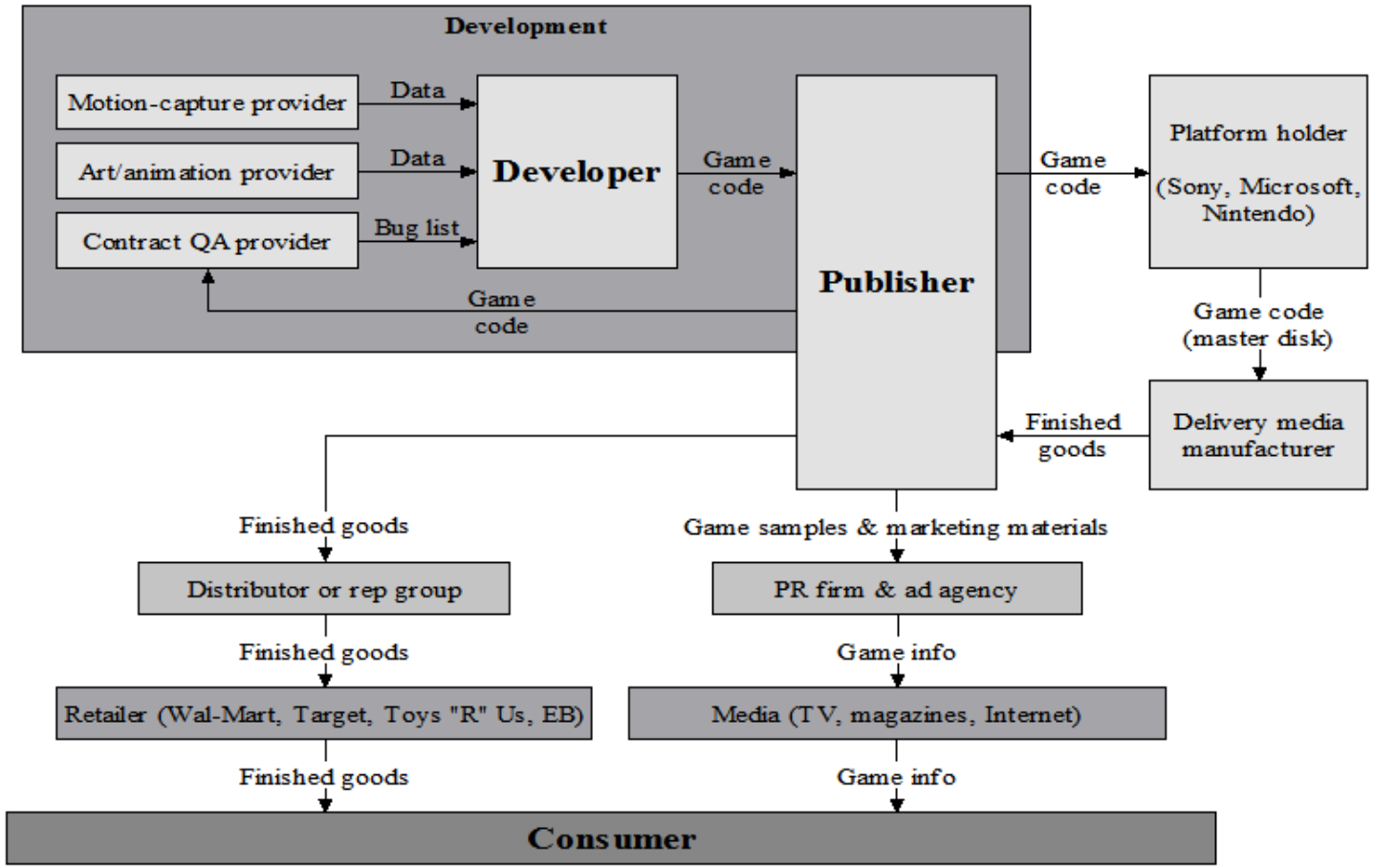
- International video game sales 2014: $81.5 billion
  - Double international film industry revenue
  - US video game sales 2014: $20.5 billion
- 1.8 billion “gamers” in the world
  - Compare to 3.0 billion people online
- Video games have been the driving force behind:
  - CPU power
  - Graphics processing power
  - Rendering and 3D projection algorithms
  - Interest in computer science/mathematics

•The Business of Games

- Developing a title for the PS4 or Xbox One
  - Costs $20 to $50 million on average
  - GTA V had a $137 million development budget
  - Plus $128 million in marketing costs
- Large game developers/publishers employ
  - Graphic artists, animators, writers
  - Vocal talent, motion capture specialists
  - Programmers, tool creators, QA testers,
  - Project managers, directors
  - Media creators, marketers, salespersons

http://www.newzoo.com/insights/top-100-countries-represent-99-6-81-5bn-global-games-market/

•Game Development Pipeline



•What is a game?

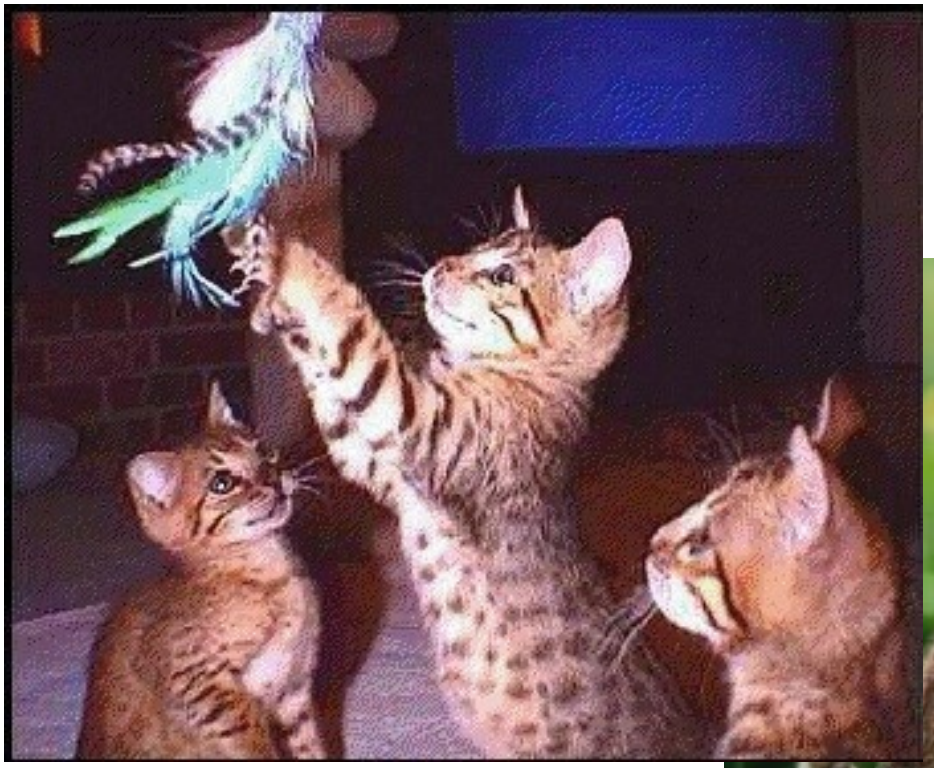
- With so much money at stake...
- Can we answer the following questions:
  - What is a game?
  - What makes a game (or anything) fun?
  - How can we create good (or successful) games?

•Game studies: Ludology vs Narratology

- Ludology: Games are experiences
  - The experiences are generated by a framework of rules
  - Analyze games in terms of abstract formal systems
  - From the Latin ludus (game) + -ology (study of).
- Narratology: Games tell stories
  - They are a form of narrative
  - But: What is the compelling story behind “tetris”?

•“Funativity”

- How, why is something fun?
- Do these kittens look like they are having fun?



•"Theory of Natural Funativity"

- Fun derives from practicing skills that aid survival
- They are a safe way to "practice" skills
- Applied to Cats:
  - Adult cats need to be able to catch small prey for food and fight for territory/mates
  - Thus kittens practice:
    - Hunting → Chasing feather, ball of string, tail
    - Fighting → Attacking each other, ball of string, your leg

•Funativity & Humans

- For most of history humans were tribal hunter-gatherers
- Fun things should be related to those skills
  - Physical: hunting, spatial reasoning
    - Shooters, sports games, hand-eye-coordination
  - Mental: pattern recognition, gathering
    - Pattern games, powerups, resources
  - Social: tribal interactions
    - High scores, head-to-head, the Sims, MMO

•1. Physical: Spatial Reasoning

- Abstract Definition: Reasoning about objects in 3D space and how they might interact (including your own body, hand-eye coordination)

•2. Mental: Pattern Recognition

- Abstract Definition: Recognizing patterns in organized sets of data, remembering chains of linked events that are significant

•3. Social

- Abstract Definition: Practicing interpersonal communication skills, competing/cooperating with others or modeling dynamics of social situations

•Concrete Components

- Sid Meier (creator of Civilization) says

  “A great game is a series of interesting and meaningful choices made by the player in pursuit of a clear and compelling goal”

- Interesting and meaningful choices
- [Intermediate rewards and punishment]
- Clear and compelling goal

Clear and compelling goal

- Player should never be “wondering what to do”
- One impossible jump, one too-difficult boss fight and the player WILL quit.

Interesting and meaningful choices

- You aren't just trying to save the princess, you are also collecting, buying, trading, completing stages.
- Choice is an illusion, only be so many paths through a game (no "infinite" content)
- But players MUST feel that their choice matter
  - If the result of winning a boss battle and losing a boss battle are the same you won't be happy
- Players want to feel that their game experience is "unique"
- Customizable characters, branching game progressions and multiple endings all help

Intermediate rewards and punishment

- Isn't the reward winning and the punishment losing? NO!!!
  - Rewards are positive reinforcement signals (auditory/visual)
  - Punishments are negative reinforcement signals (auditory/visual)
- Modern games have "rewards" about every 2 minutes (achievement unlocked!)
- You are more likely to keep playing when killed, if you're mocked in some way
- Games starting to directly condition players (addiction)?

•Narrative

- Questions:
  - What about the story?
  - Shouldn’t a game have a good story?
- The narratological view of game studies says that games should be understood as a form of storytelling ("choose your own adventure")
- Treating a game as a narrative (or including narrative as part of a game) can help us make a more compelling game, and may even be thought of as adding a "social" component

•Narrative in Games

- Ultimate goal (as with literature, and cinema) is to get the player or viewer to “suspend disbelief” and have a “real” emotional response to events that are entirely fictitious.
- Including a compelling narrative in a game can “make it incredible” (ChronoTrigger, BioShock) or simply create a series of annoying cut scenes that a player has to wade through.

•Narrative in Literature

- Rules for narrative in literature have been around since the time of the Greeks (Aristotle's Poetics).
- Questions to ask:
  1) Who is telling the story?
  2) What is the conflict?
  3) With whom is the player meant to identify?
  4) What do you want the player to feel?
- Example game

•Narrative in Film

- Modern games have far more in common with film (cinematography) then with regular literature
- Cinema also has a lexicon of well established rules regarding the creation of compelling narrative

  1) Don't break the narrative plane
  2) Don't break the narrative chain
  3) Use the camera to frame action*
  4) Use the camera to immerse the viewer*

  (*Note: What's unique about games is that you always have perfect camera, light, etc.)

•Mechanics, Dynamics & Aesthetics (MDA)

- MDA is a game development paradigm designed to help developers make the most out of a game idea, and proceed efficiently through the complex process of bringing a game to market
- MDA is one of many development paradigms that are rigidly used by large game development companies

•Mechanics

- Before a single line of code is written the mechanics that will be used by the game should be well thought out and documented
- This includes:
  - The programming language
  - The programming libraries, engines, tools
  - The hardware required/available
  - The logical programming components
  - The storage/retrieval/initialization methods

•Dynamics

- Before a single line of code is written the dynamics that will be used by the game should be well thought out and documented. This is the “ludological” part of MDA. All objects and axioms need to be detailed!
- This includes:
  - The domain of the game
  - The players in the game
  - The rules of the game
  - The objects in the game

•Aesthetics

- Before a single line of code is written the aesthetics that will be used by the game should be well thought out and documented. This is the "narratological" part of MDA
- The "art bible" which should contain every detail of the "look" of the game will come out of this development area
- This includes:
  - Color Palette
  - Physical looks for all players
  - Lighting plots, schemes, etc.

•Genres

- MDA also gives us a way to classify games
- Group by mechanics:
  - IPhone game, C++ game, Quake Engine
- Group by dynamics:
  - Shooter, Strategy, Role-playing game (RPG), Massively multiplayer online RPG (MMORPG)
- Group by aesthetics:
  - Campy, Cartoony, Fantasy, Sci-Fi, Horror Survival

Summary

- Theory of Funativity
  - Spatial, mental, or social challenge
- Concrete rules (Sid Meier)
  - Goals, choices, punishments/rewards
- Narrative
  - How do you want the player to feel?
- Methodology like MDA
  - Mechanics, dynamics, aesthetics

CISC 1600 Lecture 3.3
Game state and math

Topics:

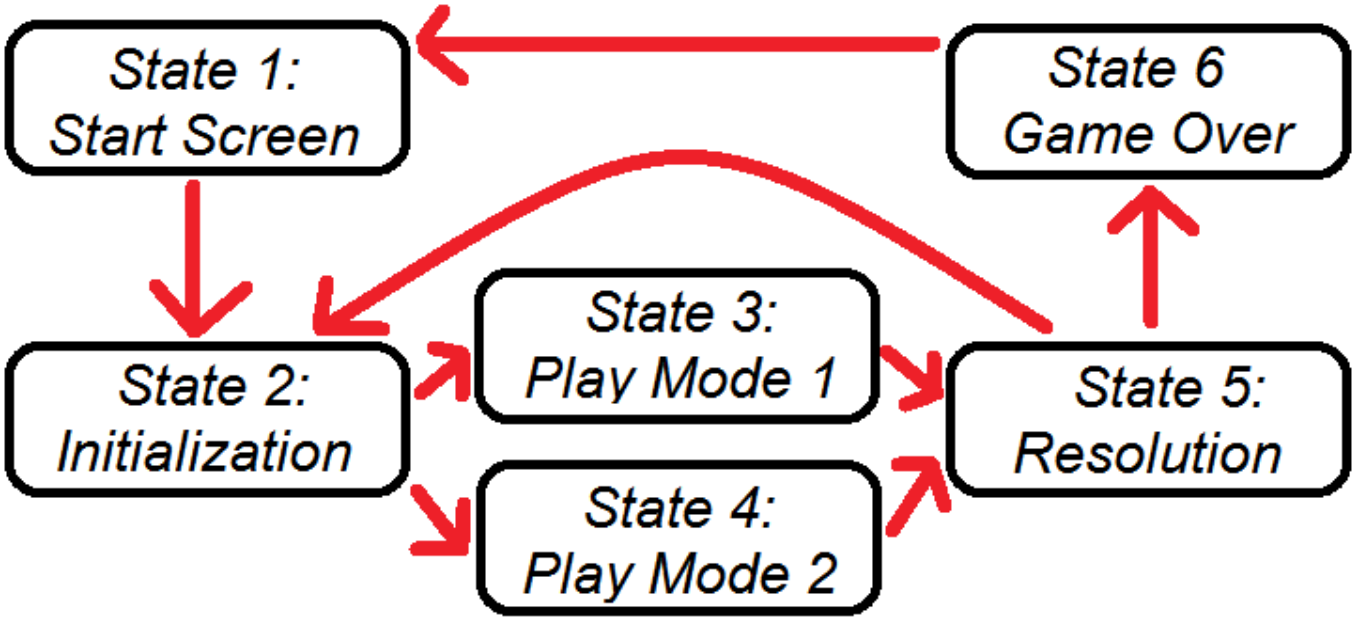
Game state and object state

Game physics

Collision detection

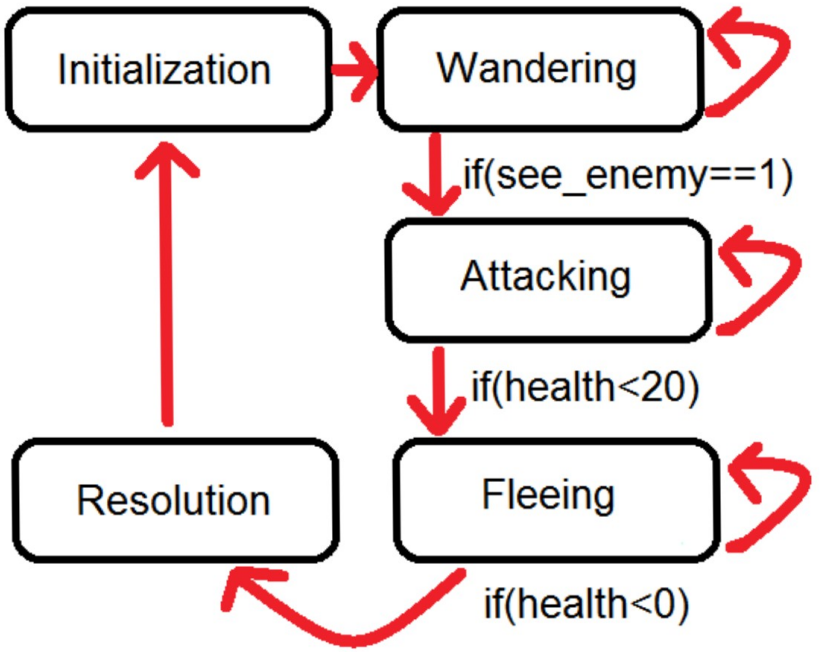
•Game state controls global behavior

- All games consist of a sequence of states.
- Each state is characterized by a combination of visual, audio and/or animation effects
- As well as a set of rules that are being applied.



•Object state controls local behavior

- Objects in the game proceed through their own states as well
  - Same representation, different meaning
- These states are defined by the game state as well as local conditions for that object (health, position, etc.)

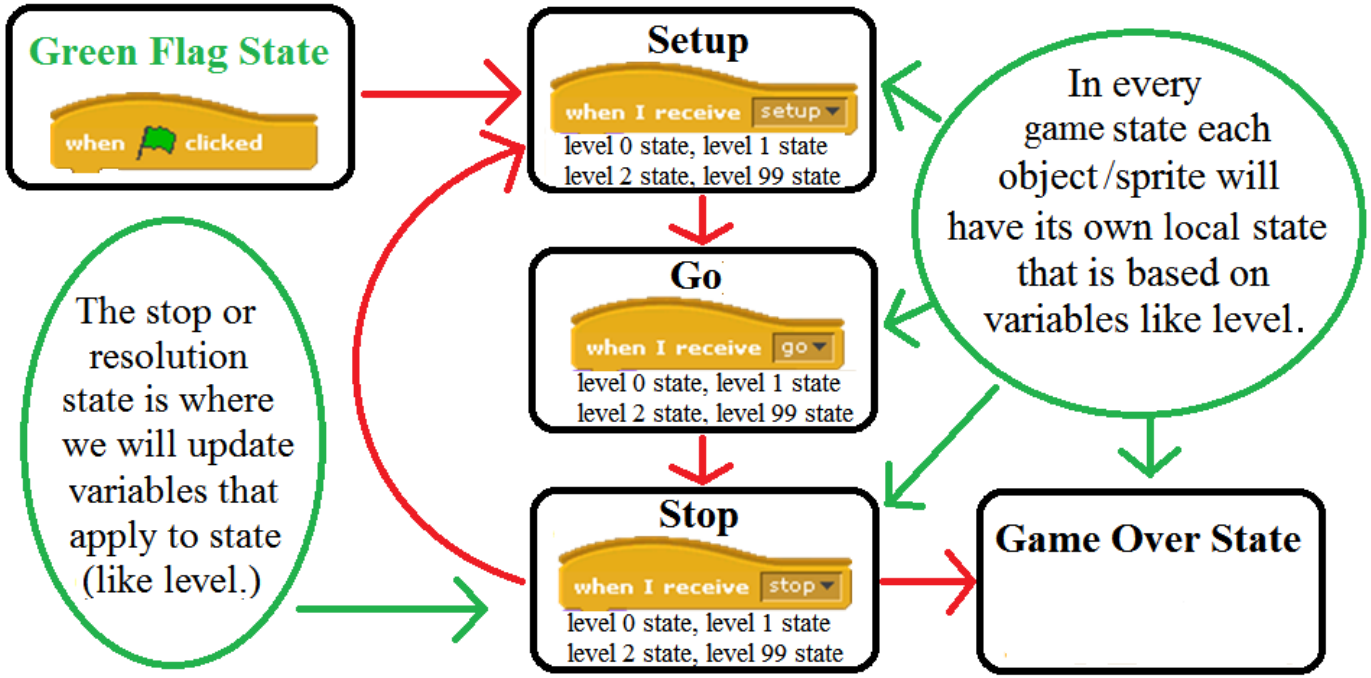


State machines are a general way to represent this

- A state machine is a collection of states and legal transitions between them
- The machine is in one state at a time (at all times)
- The machine can transition between states based on various events
- The behavior of the machine only depends on its current state, not any previous state
- Examples: vending machine, combination lock

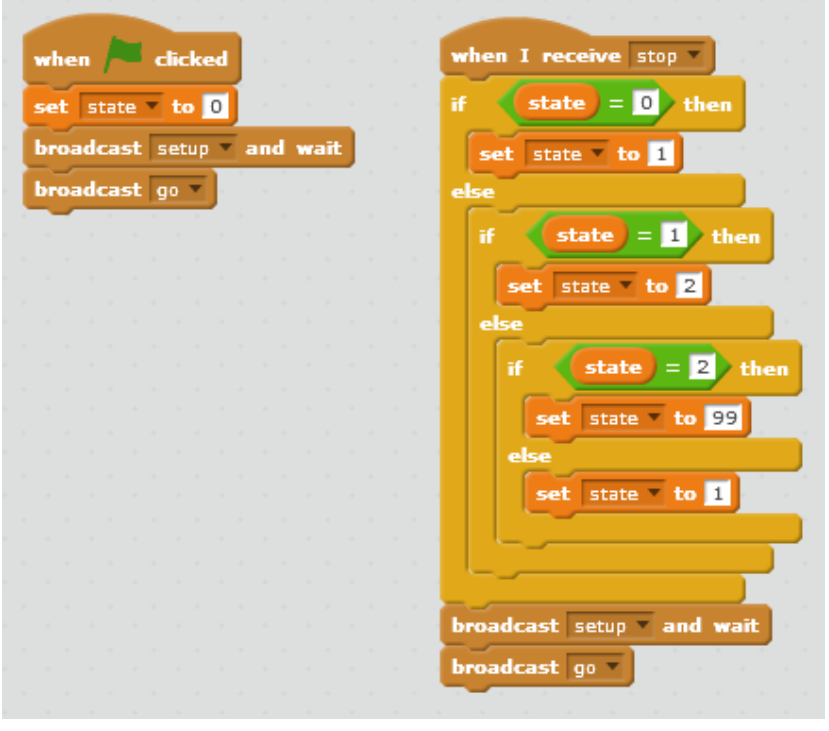
Implementing states in Scratch

- Create a global variable to store the game state
- Create a local variable for each sprite to store its object state
- Broadcast 3 messages (“phases”) for each game state:
  - Setup, go, stop



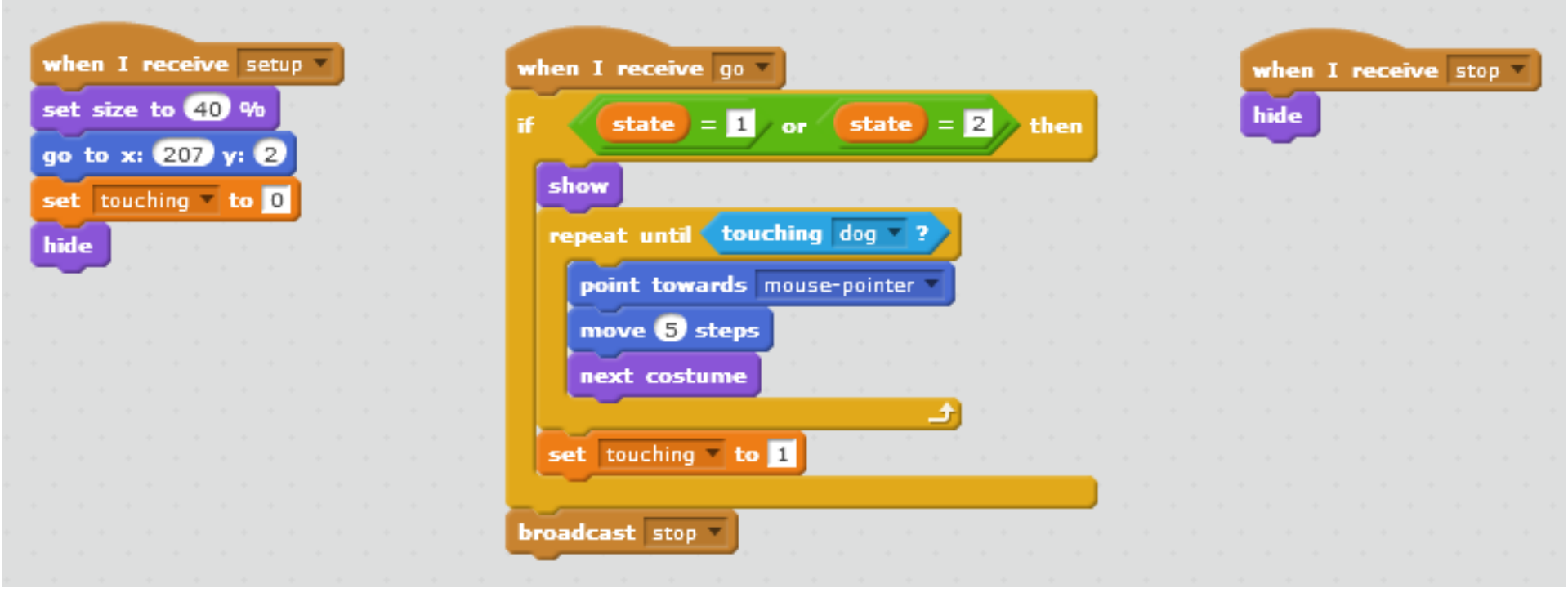
Someone needs to "own" game state

- The stage is a good sprite to "own" the game state
- Initialize it, transition between game states
- Be careful about "passing" it to another sprite
  - Sometimes necessary



•Using game state in sprites

- In this case, all normal sprites have just 3 scripts
- Each script handles one of the messages
- Each script has many “if” blocks to change behavior based on the game and object state

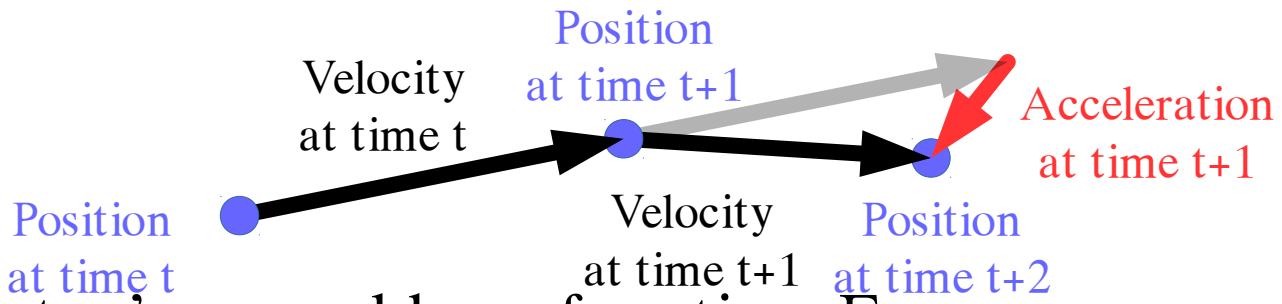


•Game Mathematics

- "Game Mathematics" encompasses many areas of math: Geometry, trigonometry, calculus, linear algebra (vectors, matrices), etc.
- Libraries for graphics (2D, 3D) and games exist, and help simplify development
- But they can’t be relied on to do everything.

Game physics: Position, velocity, acceleration

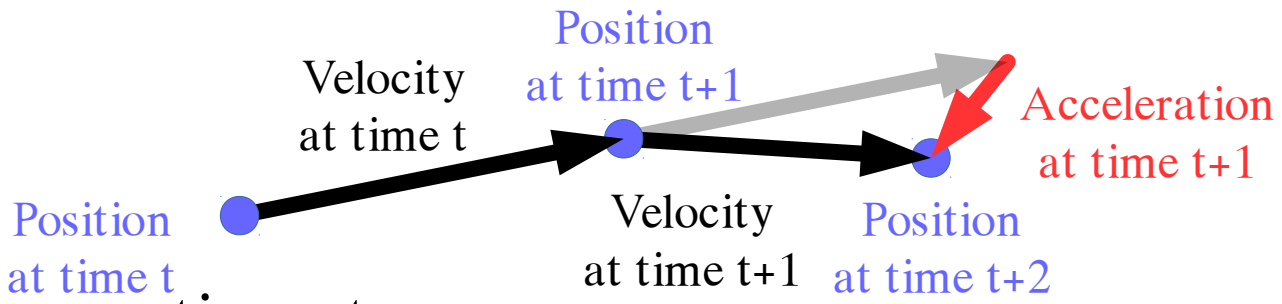
- Need three variable groups to realistically simulate motion
  - Position: 2D or 3D coordinate of a point's location
  - Velocity: rate of change of position over time
  - Acceleration: rate of change of velocity over time



- Newton's second law of motion: F = ma
  - Force = mass * acceleration
  - Pushing on something accelerates it

Game physics: Position, velocity, acceleration

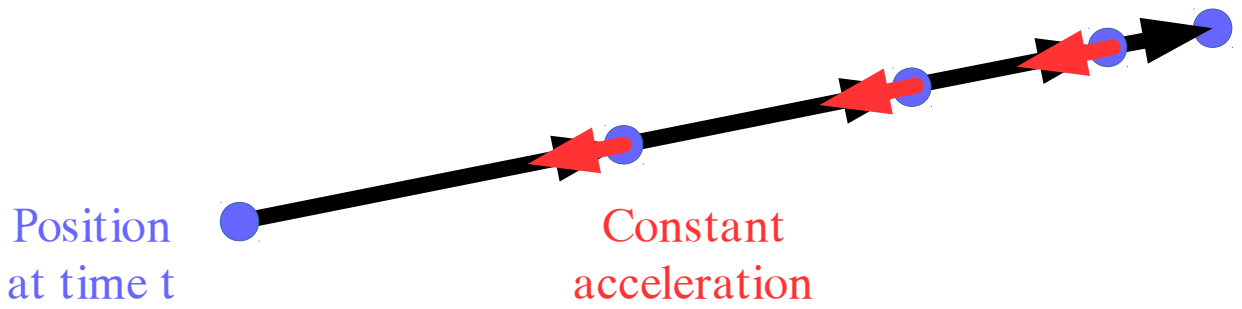
- Need three variable groups to realistically simulate motion
  - Position: 2D or 3D coordinate of a point's location
  - Velocity: rate of change of position over time
  - Acceleration: rate of change of velocity over time



- At every time step
  - Position = position + velocity
  - Velocity = velocity + acceleration
  - Acceleration = force / mass

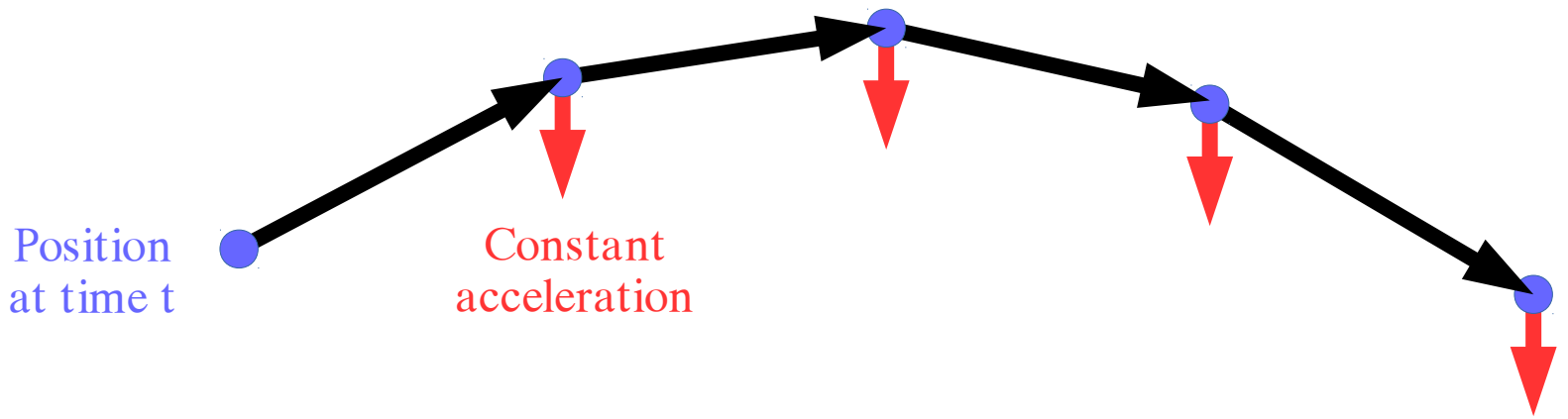
Game physics: friction

- Friction is a force against the direction of motion
  - Of a magnitude independent of the velocity



Game physics: gravity

- Gravity is a constant acceleration downwards
- Simulate projectiles, jumping, explosions, etc.



•Game physics: collisions

- Bouncing off an immovable object reverses the part of the velocity perpendicular to that object