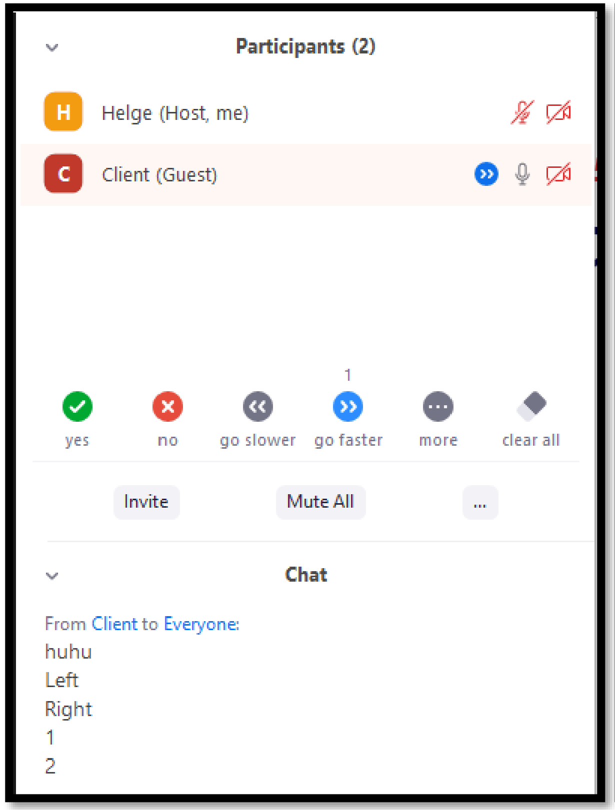
CPSC 427
Video Game Programming

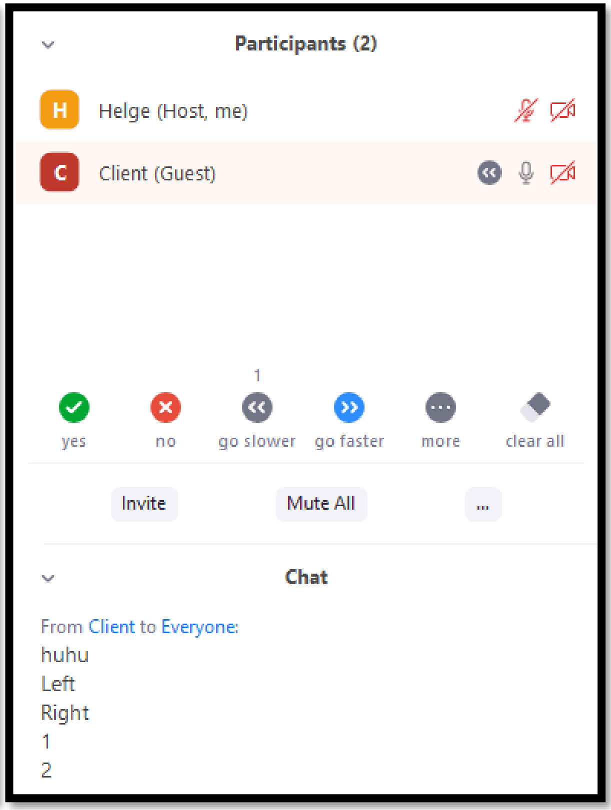
Game Play and AI

Helge Rhodin

Read the zoom chat

- ***Capture the screen***
  - *https://github.com/smasherprog/screen_capture_lite*
- ***Search for the zoom window***
- ***Check for colored symbol***
- red, green, gray, blue?
  - *only need to read a few pixels*
    - its fast!



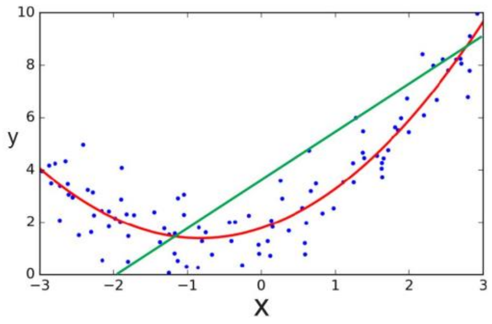


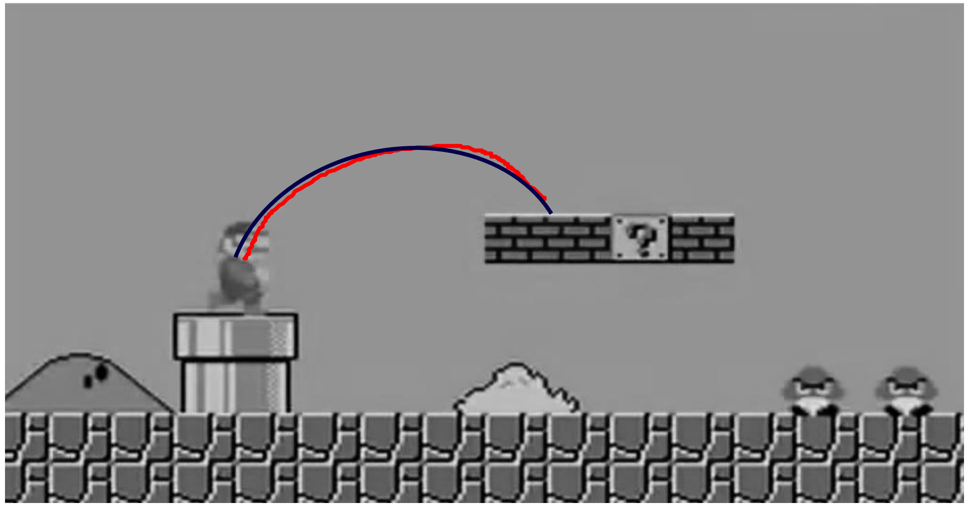
- ***Recognize numbers?***
- only 10 different ones, brute force?

Mouse gestures

Regression

- ***least squares fit***
  - linear, polynomial, and other parametric functions



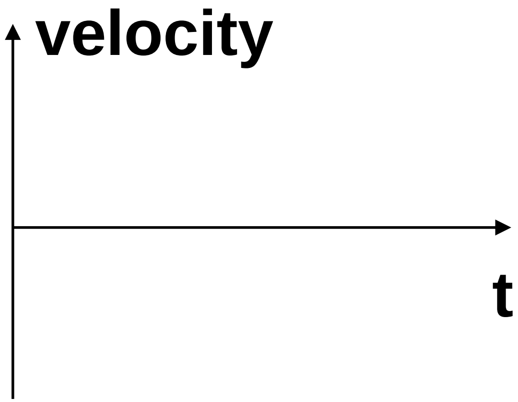


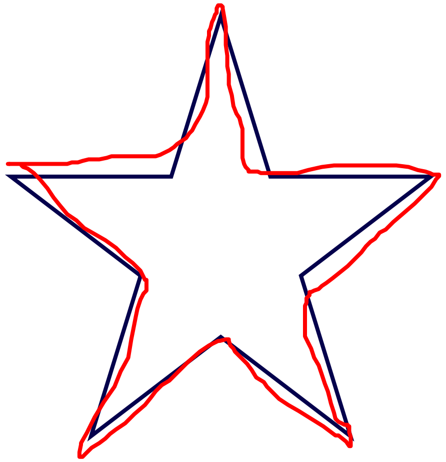
Search

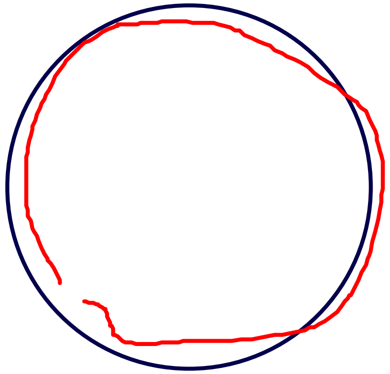
- brute force?
- binary search?

Detection

- key events
- pattern matching







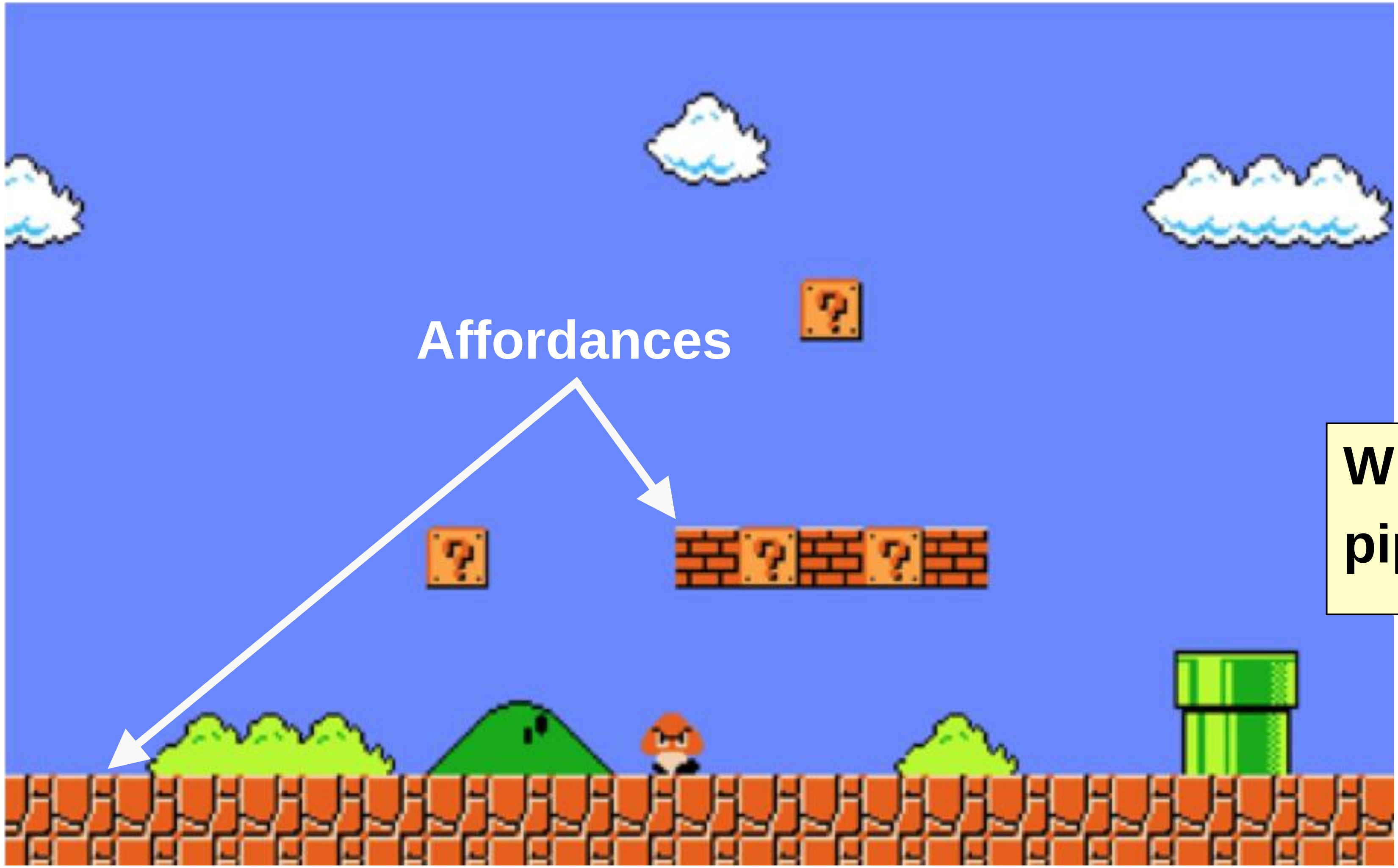
Connection to Game Design

- **Impact of design on ease of use & engagement**

In Wind Waker, the direction Link looked indicated to the player something of interest was there

- **Design applications & philosophies are interconnected**

Example of Affordances in Games



What does the pipe afford?

Users

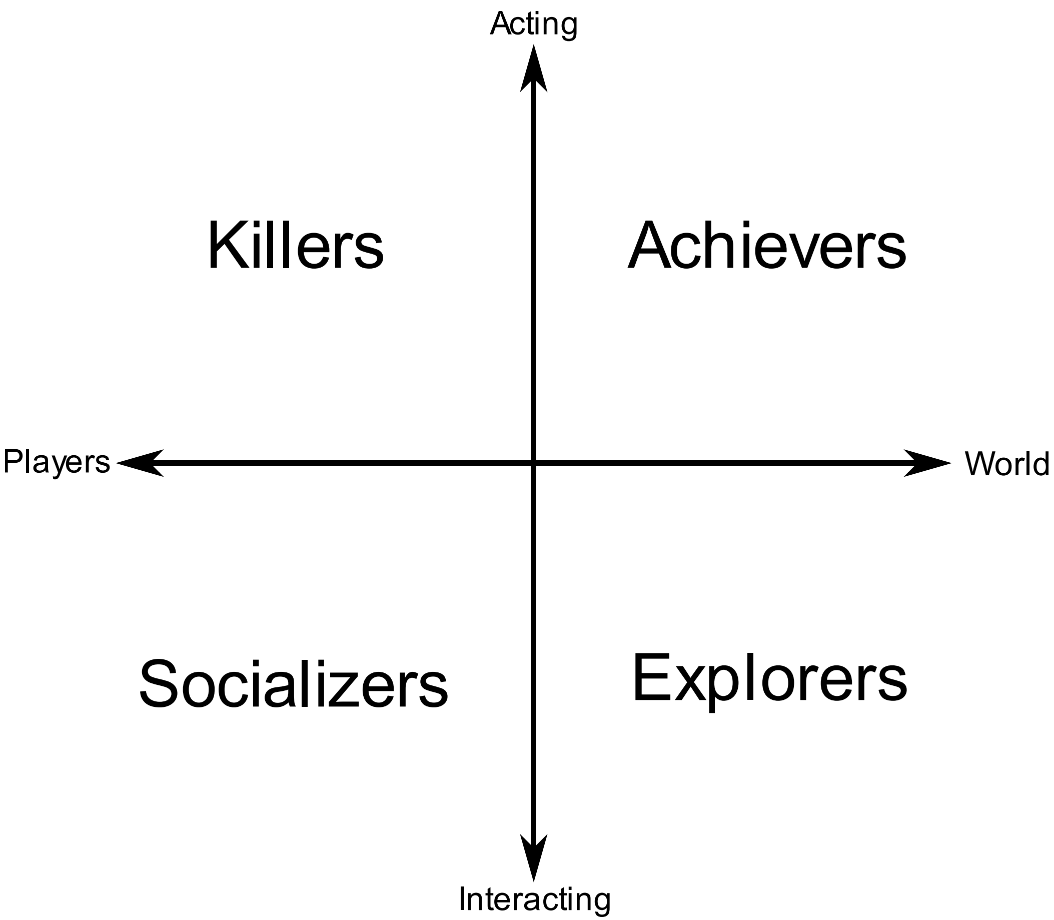
- **Who are the players?**
  - ***Age: Children, adults, university students***
  - ***Culture***

- **Where will they be playing?**
  - ***Commuting, at home, remotely***

- **What do they need or want?**
  - ***Fulfilling plot, relaxing play***

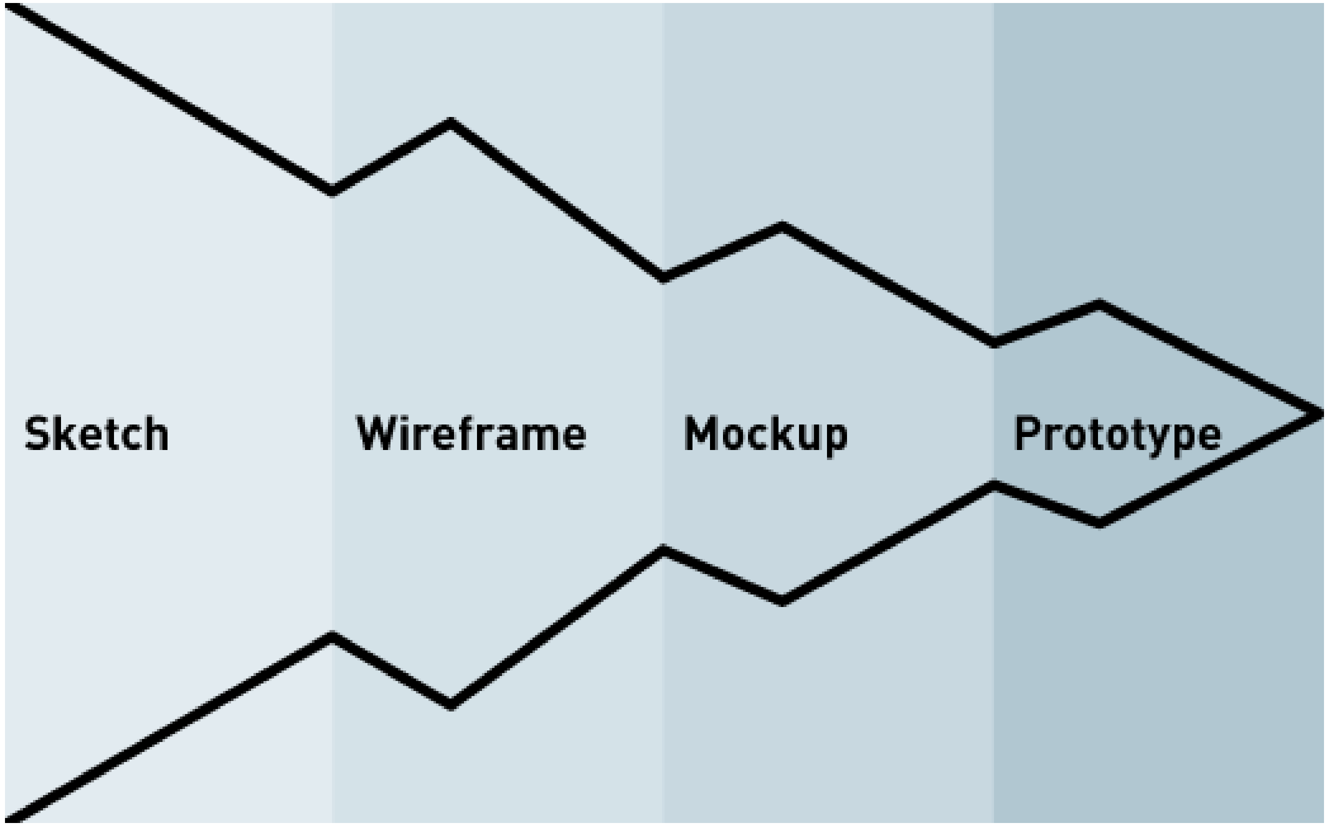
What Motivates Users?

- **Work has been done to identify player types**
- **Users can be classified by preference for interacting/acting with/on others/the world**
- **The four classifications tell us what motivates each player type**



The Design Process

Brainstorming



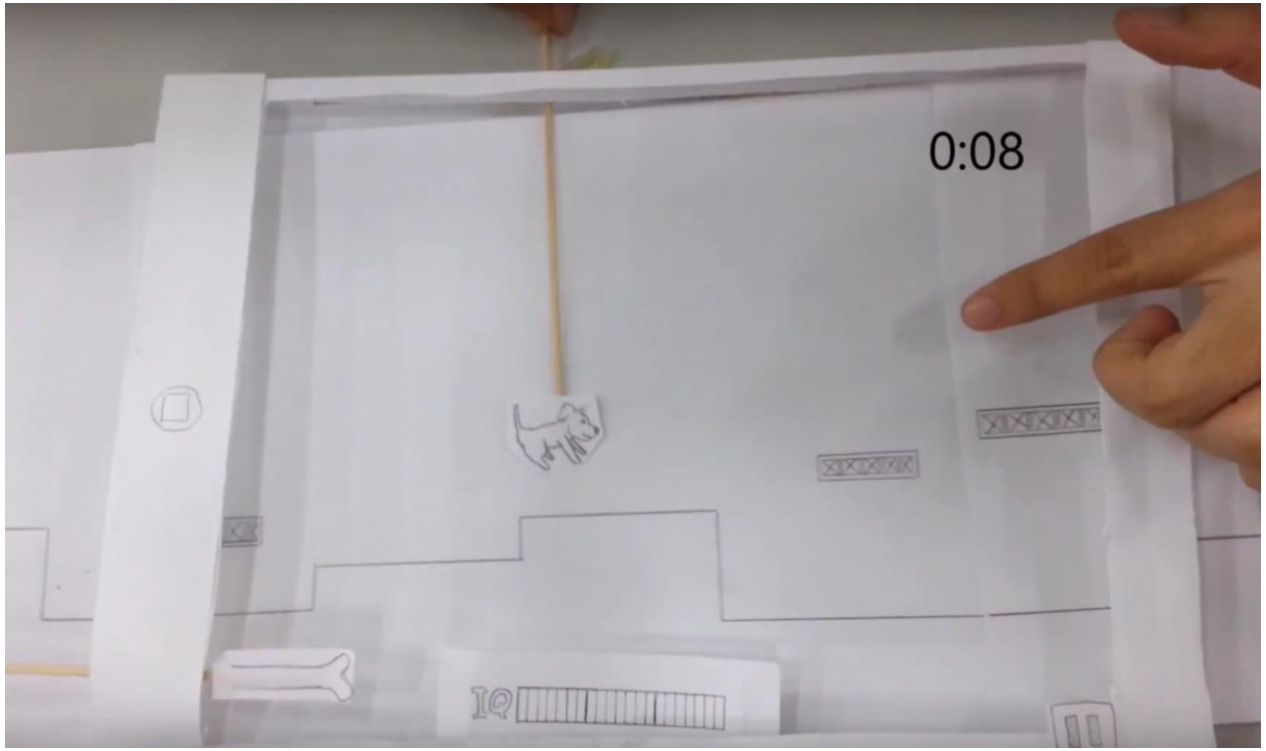
Release

Low fidelity prototyping

High fidelity prototyping

Low Fidelity Prototyping

- Used for **early** stages of design
  - *Quick & cheap to deploy*
  - *Easy to test*
- Iterate on **story** and **core gameplay mechanics**
- **Sketches** are a great way to start designing



Testing Low Fidelity Prototypes

- Don’t commit to one approach, design a few prototypes & compare
- Invite someone to try them out
- Try to drill down on feedback
  - *If they just say it’s “fun”, ask why?*

Fail Early, Fail Often, and Iterate on Feedback

- **Designing something that people will use is both an art & a science**
  - ***follow established principles***
  - ***Iteration is how you make it better***

- **Early feedback ensures design meets users' needs**

- **Throwing around ideas is quick**
  - ***Fixing a bad design is expensive***

- **No idea is perfect the first time around**

Medium Fidelity Prototyping

- Use medium fidelity prototyping for the early to middle stages of design
  - *Identify questions before coding*
  - *Be selective with what gets built*
  - *Get it right in black and white first*

- Iterate on tone & feel of game
  - *Supplementary game mechanics*
  - *Rough visuals & audio*
  - *Feedback*

Greyboxing

- Greyboxing blocks out all elements as shapes to test gameplay

CPSC 427
Video Game Programming

Game Play and AI

Helge Rhodin

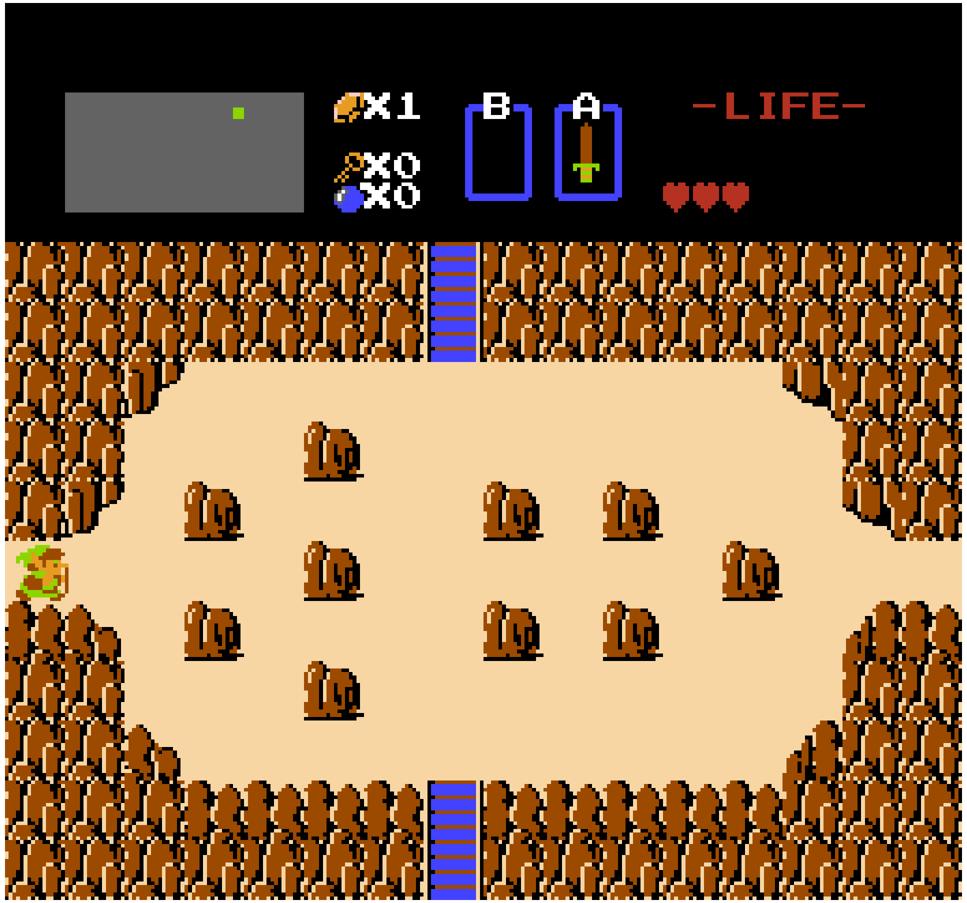
Invited Talk Schedule

Tuesday, March 2., 5-6 pm	**Yggy King** (Blackbird Interactive)	**ECS**
Tuesday, March 9., 5-6 pm	**Craig Peters** (EA)	**Debugging**
Tuesday, March 16., 5-6 pm	**Ben** (Brace Yourself Games)	**UI development**
Tuesday, March 23., 5-6 pm	TBD (Skybox)	**ECS and multi-threading**
Tuesday, March 30., 5-6 pm	**Dinos** (Charm Games)	**Moving &rendering in VR**

Nvidia: RTX and raytracing (working on it)

ECS examples – entity, component, or system?

World grid

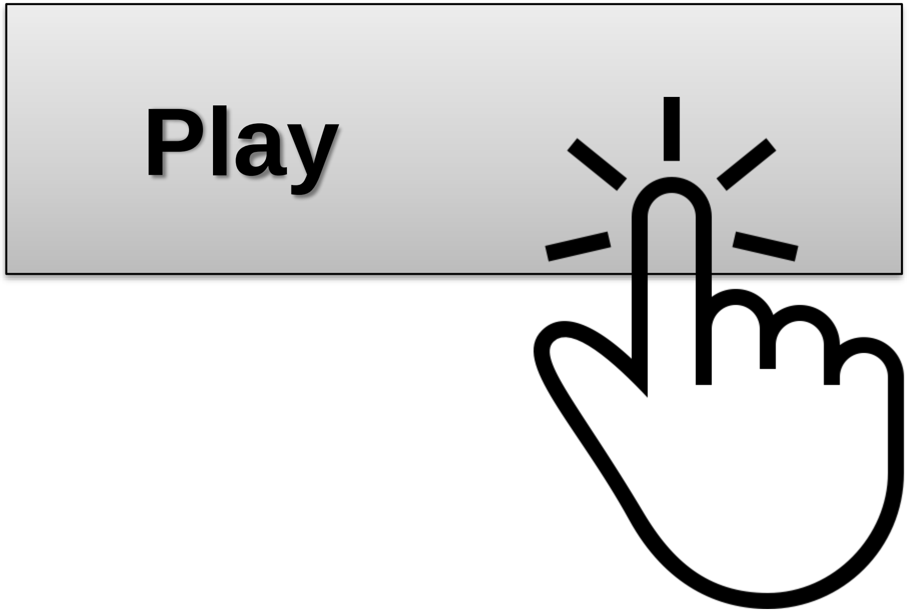



```
enum Tile {empty, wall,
           ladder, rock, …};
```

1	1	1	2	1	1	1
1	0	0	0	0	0	1
0	0	3	0	3	0	0
1	0	0	0	0	0	1
1	1	1	2	1	1	1

Menu item

- ***component, system, entity?***



Level Loading with JSON

Libraries:

- ***https://sourceforge.net/projects/libjson/***
- ***https://github.com/nlohmann/jso***
- ***others?***

Loading Entities and Components

- Outer list of entities
- Inner list of components
- Create a factory that instantiates each component type
- Equip components with toJSON(…) and fromJSON(…) functions

```
“entities": [
  {
    "position": {
      "x": -1.7193701,
      "y": -0.09165986
    },
    "velocity": {
      "x": 0,
      "y": 0
    },
    "color": {
      "x": 0.453125,
      "y": 0.453125,
      "z": 0.453125
    },
    "type": “Water Animal"
  },

  {
      "position": {
        "x": 2.2221813,
        "y": -1.2671415
      },
      "velocity": {
        "x": 0,
        "y": 1
      },
      "radius": 0.9300000000000006,
      "color": {
        "x": 0.40625,
        "y": 0.40625,
        "z": 0.40625
      },
      "type": ”Land Animal”
    }
  ]
```

Factory from JSON

Factory:

```
void ComponentfromJson(Entity e, JsonObject json)
{
  if(str1.compare(“Motion") != 0) {
     auto motion = Motion::fromJson(json);
     ECS::registry->insert(e, motion);
  }
  else if(str1.compare(“Salmon") != 0)
     auto component = Motion::fromJson(json);
     ECS::registry->insert(e, component);
  }
  …
}
```

Component from JSON

Component to/from:

```
class Vector2D
{
    float x,y;
    public:
    JsonObject toJson()
    {
        JsonObject json = Json.object();
        json.add("x", x);
        json.add("y", y);
        return json;
    }

    static Vector2D fromJson(JsonObject json)
    {
        double x = json.getFloat("x", 0.0f);
        double y = json.getFloat("y", 0.0f);
        return Vector2D(x,y);
    }
}
```

CPSC 427
Video Game Programming

State machines

Helge Rhodin

Gameplay

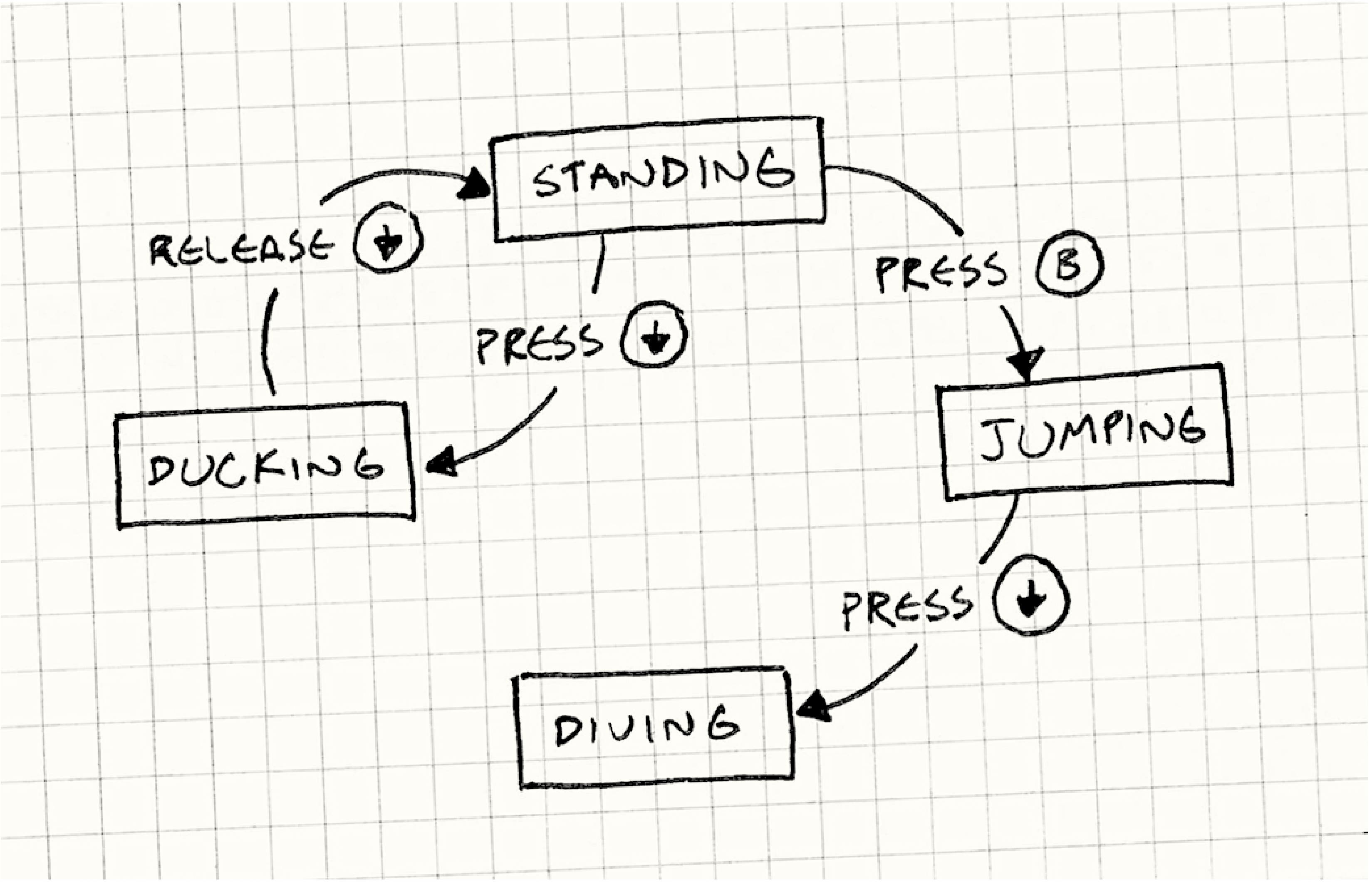
```
// start
if (!walking && wantToWalk)
{
    PlayAnim(StartAnim);
    walking = true;
}

// walk loop
if (IsPlaying(StartAnim) && IsAtEndOfAnim())
{
    PlayAnim(WalkLoopAnim);
}

// stop
if (walking && !wantToWalk)
{
    PlayAnim(StopAnim);
    walking = false;
}
```

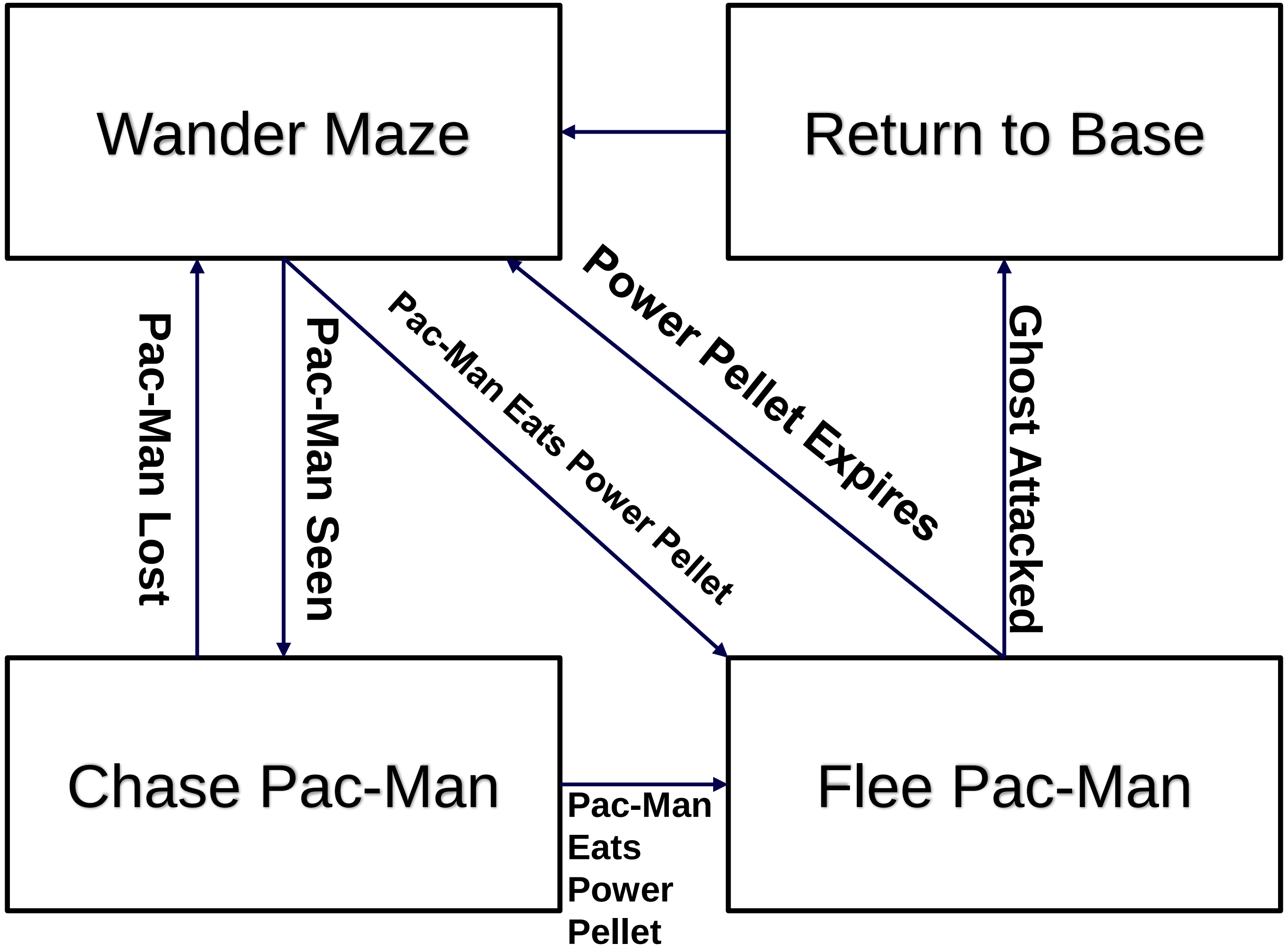
From http://twvideo01.ubm-us.net/o1/vault/gdc2016/Presentations/Clavet_Simon_MotionMatching.pdf

Finite State Machines: States + Transitions



FSM Example: Pac-Man Ghosts

FSM Example: Pac-Man Ghosts



Ghost AI in PAC-MAN

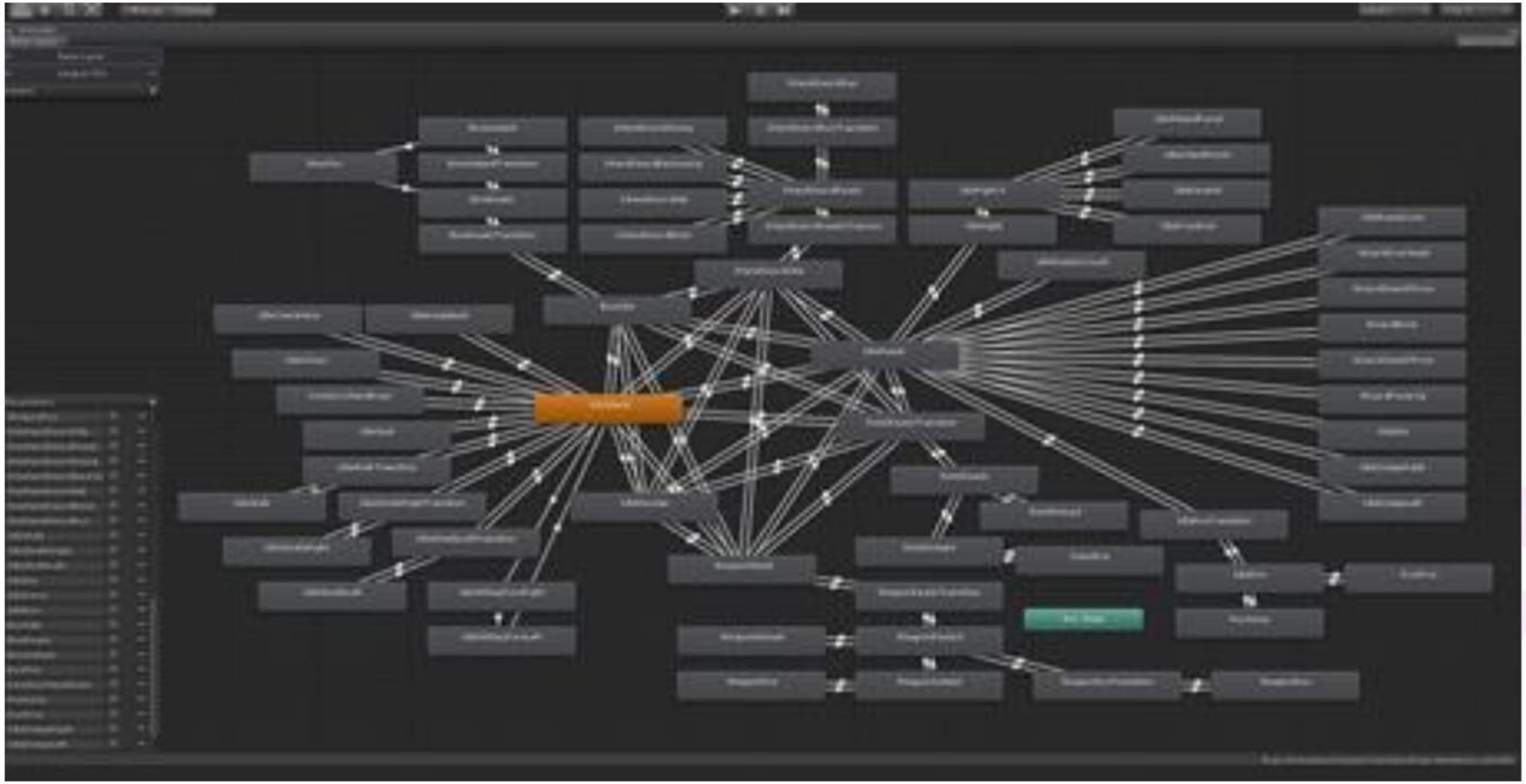
Is the AI for Pac-Man basic?

- chase or run.
- binary state machine?
- Toru Iwatani, designer of Pac-Man explained:
  "wanted each ghostly enemy to have a specific character and its own particular movements, so they weren't all just chasing after Pac-Man... which would have been tiresome and flat."
- the four ghosts have four different behaviors
  - different target points in relation to Pac-Man or the maze
  - attack phases increase with player progress
  - More details: http://tinyurl.com/238l7km

Finite State Machines (FSMs)

- ***Each frame:***
  - Something (the player, an enemy) does something in its state
  - It checks if it needs to transition to a new state
    - *If so, it does so for the next iteration*
    - *If not, it stays in the same state*
- ***Applications***
  - Managing input
  - Managing player state
  - Simple AI for entities / objects / monsters etc.

FSMs: States + Transitions



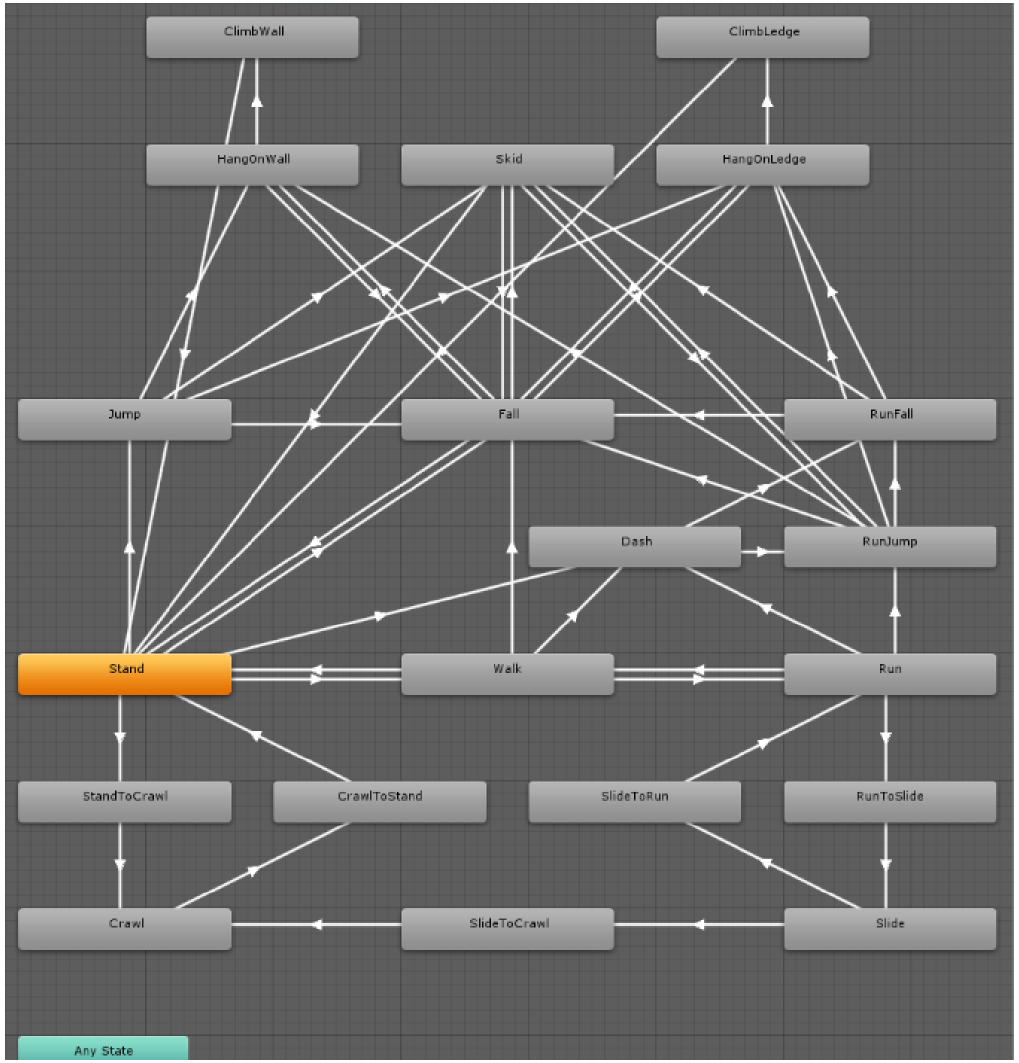
From http://twvideo01.ubm-us.net/o1/vault/gdc2016/Presentations/Clavet_Simon_MotionMatching.pdf

FSMs: States + Transitions

```
if (speed > 3.0f)
{
    PlayAnim(RunAnim);
}
else if (speed > 0.0f)
{
    PlayAnim(WalkAnim);
}
else
{
    PlayAnim(IdleAnim);
}
```

From http://twvideo01.ubm-us.net/o1/vault/gdc2016/Presentations/Clavet_Simon_MotionMatching.pdf

FSMs: Failure to Scale



No way to do long-term planning

No way to ask “How do I get here from there?”

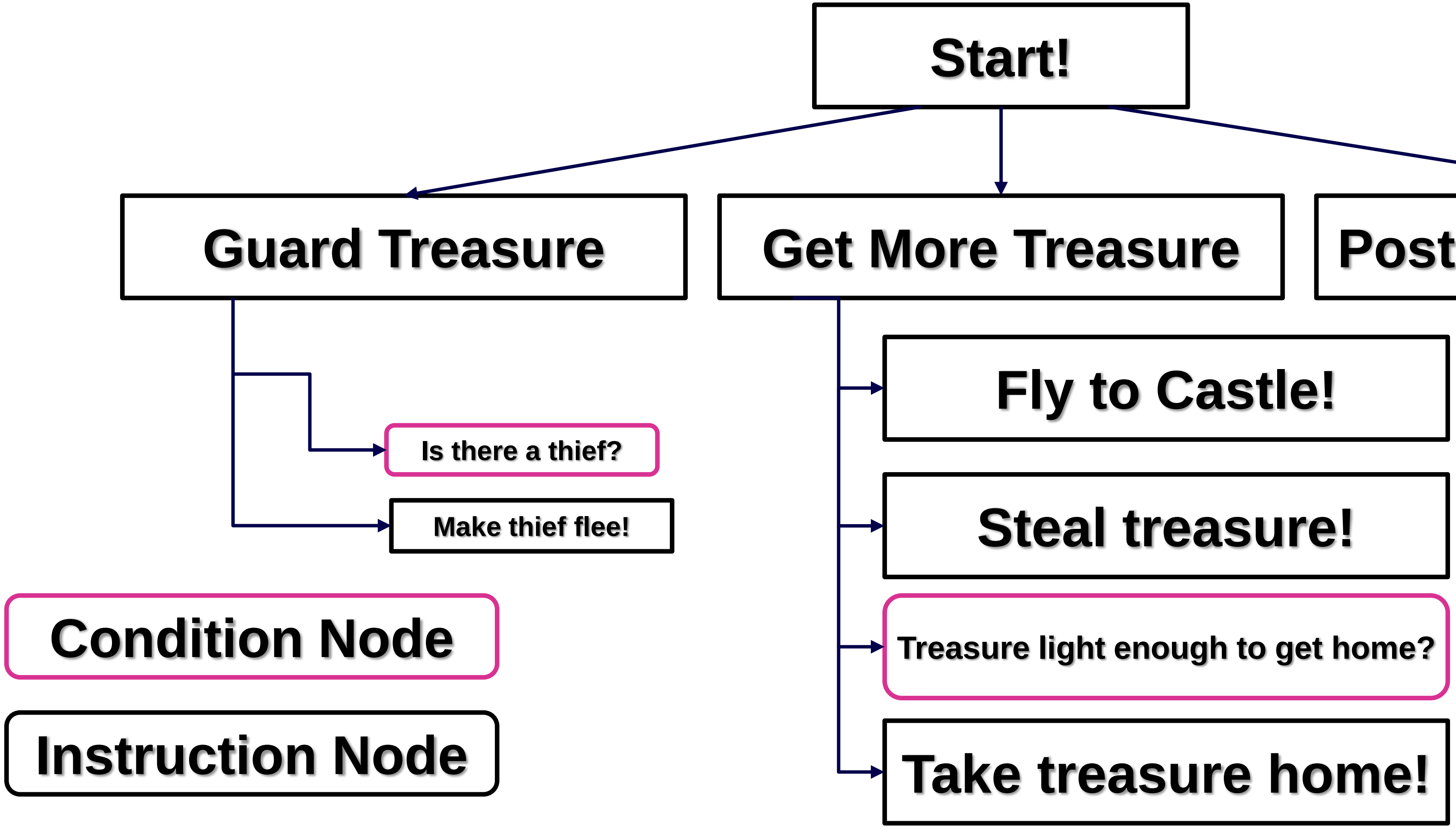
No way to reason about long-term goals

FSMs can get large and hard to follow

Can’t generalize for larger games

From http://twvideo01.ubm-us.net/o1/vault/gdc2016/Presentations/Clavet_Simon_MotionMatching.pdf

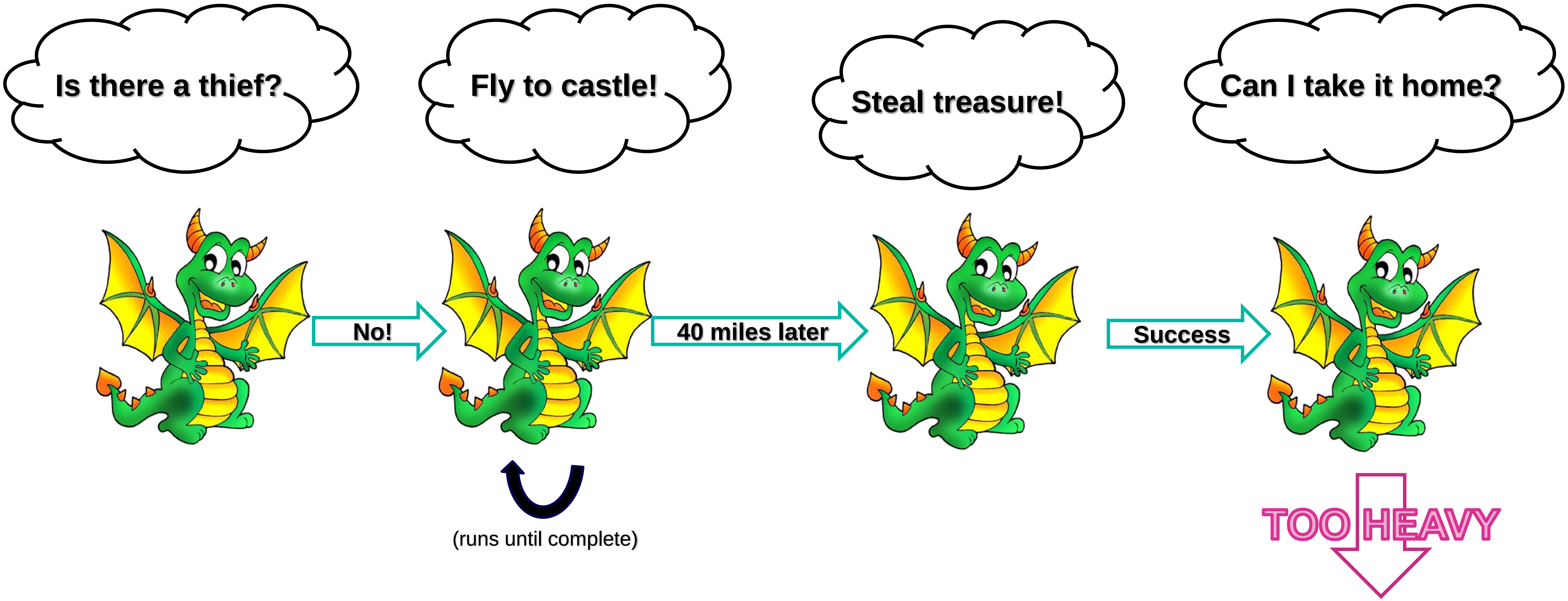
Behaviour Trees:
How To Simulate Your Dragon



Behaviour Trees

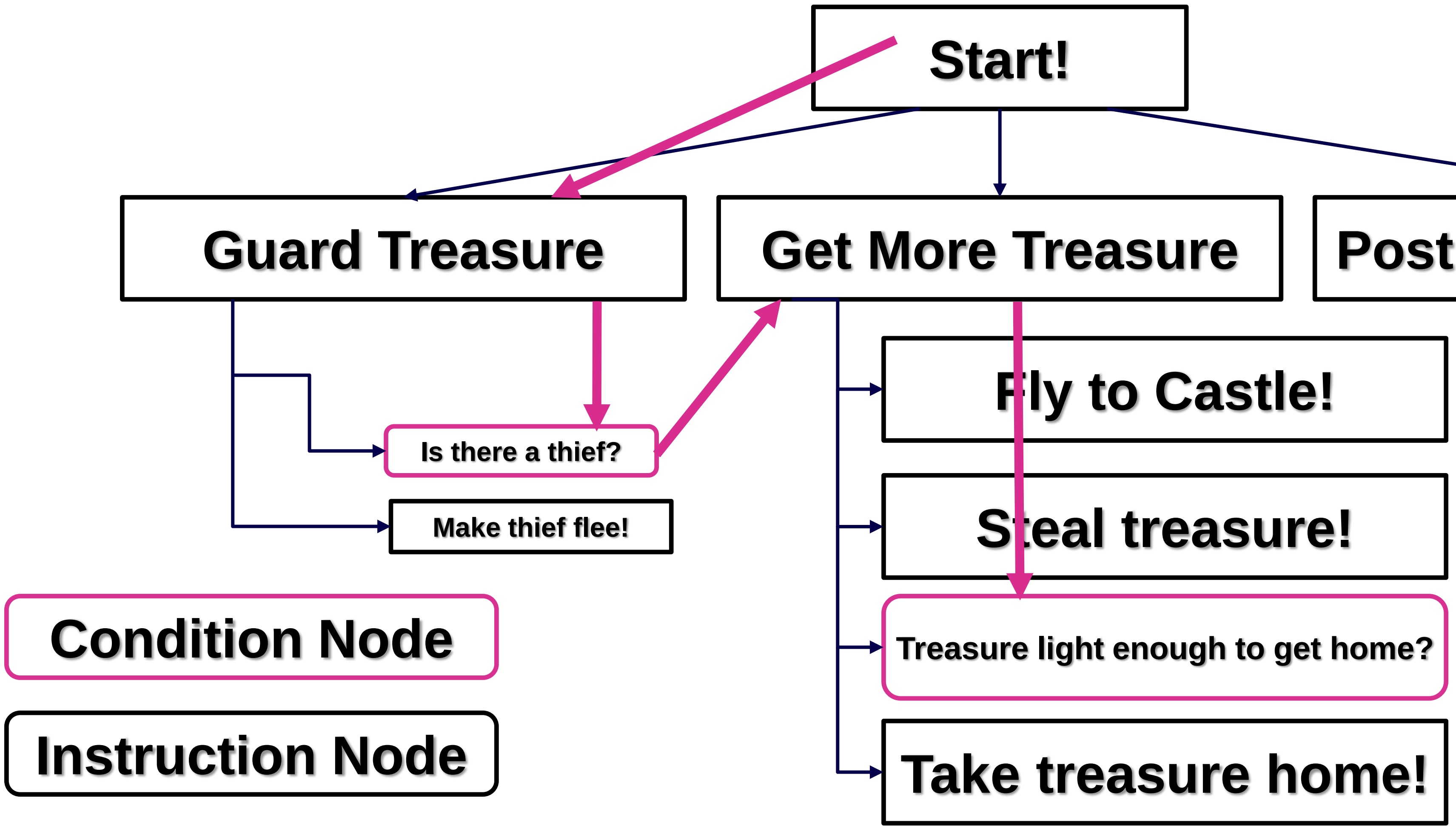
- flow of decision making of an AI agent
- ***Each frame:***
  - Visit a node
  - See if any **higher priority** nodes now run
    - *If so, execute them instead*
  - See if my currently running node fails
    - *If so, return to the root of the behaviour tree! Start again!*
  - See if the currently running node is done
    - *If so, run the **lower priority** node in the current branch of the tree*

Start!

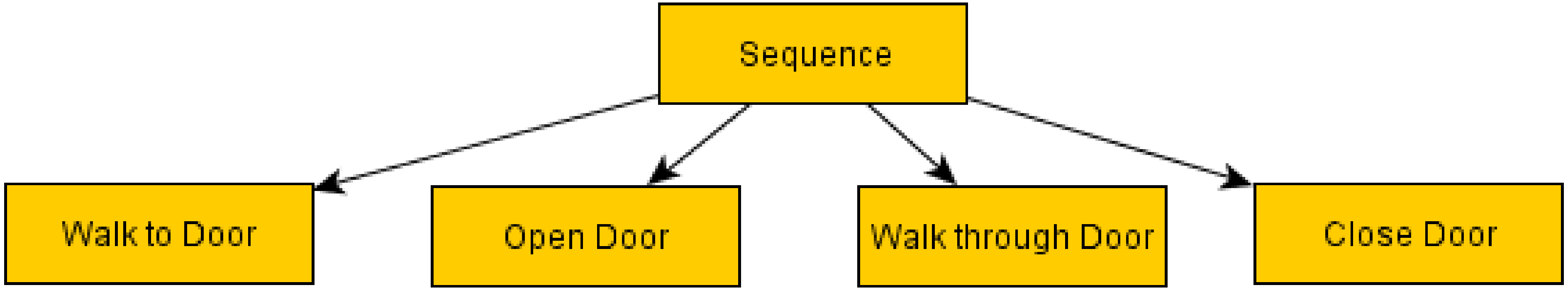


Is there a thief?
Fly to castle!
Steal treasure!
Can I take it home?
No!
40 miles later
Success
(runs until complete)
TOO HEAVY

Behaviour Trees:
How To Simulate Your Dragon

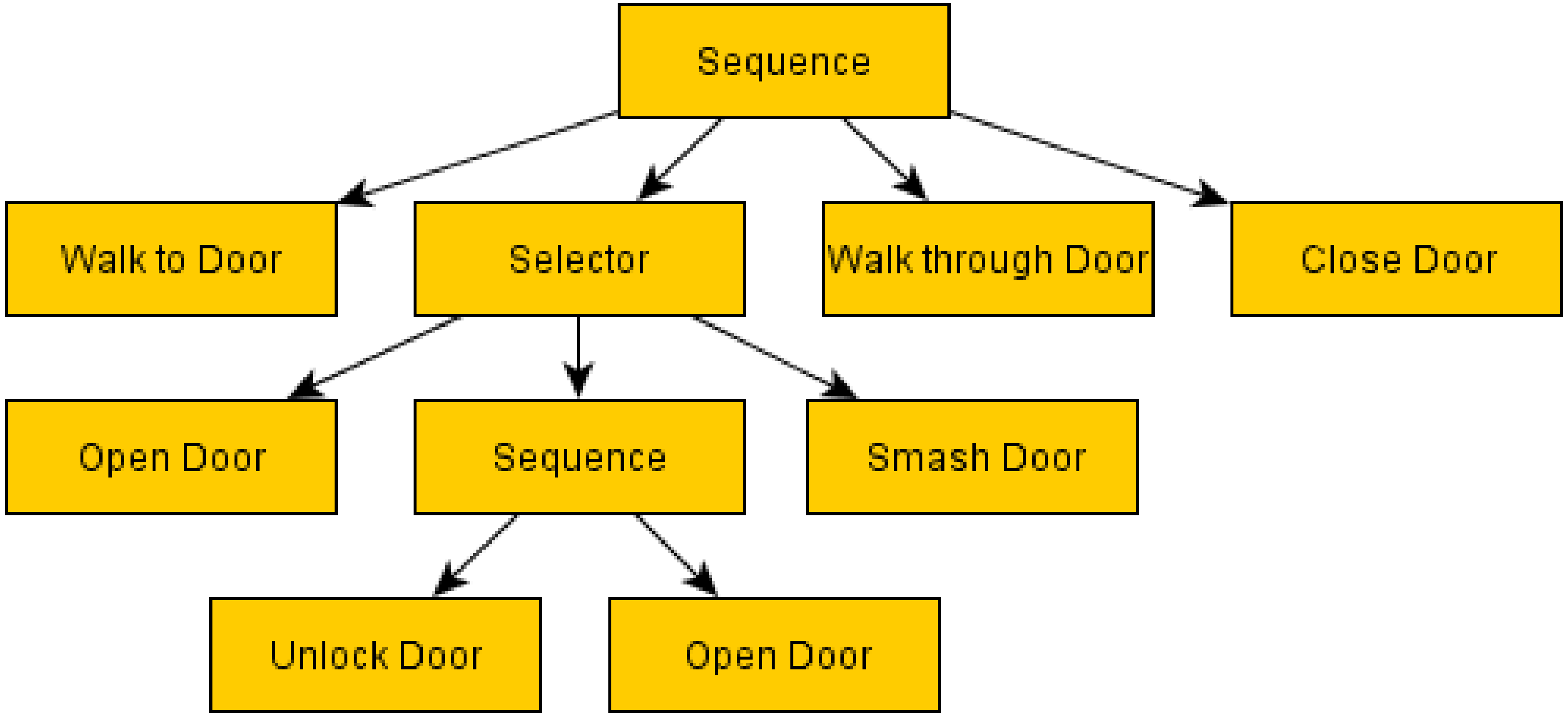


Schematic examples



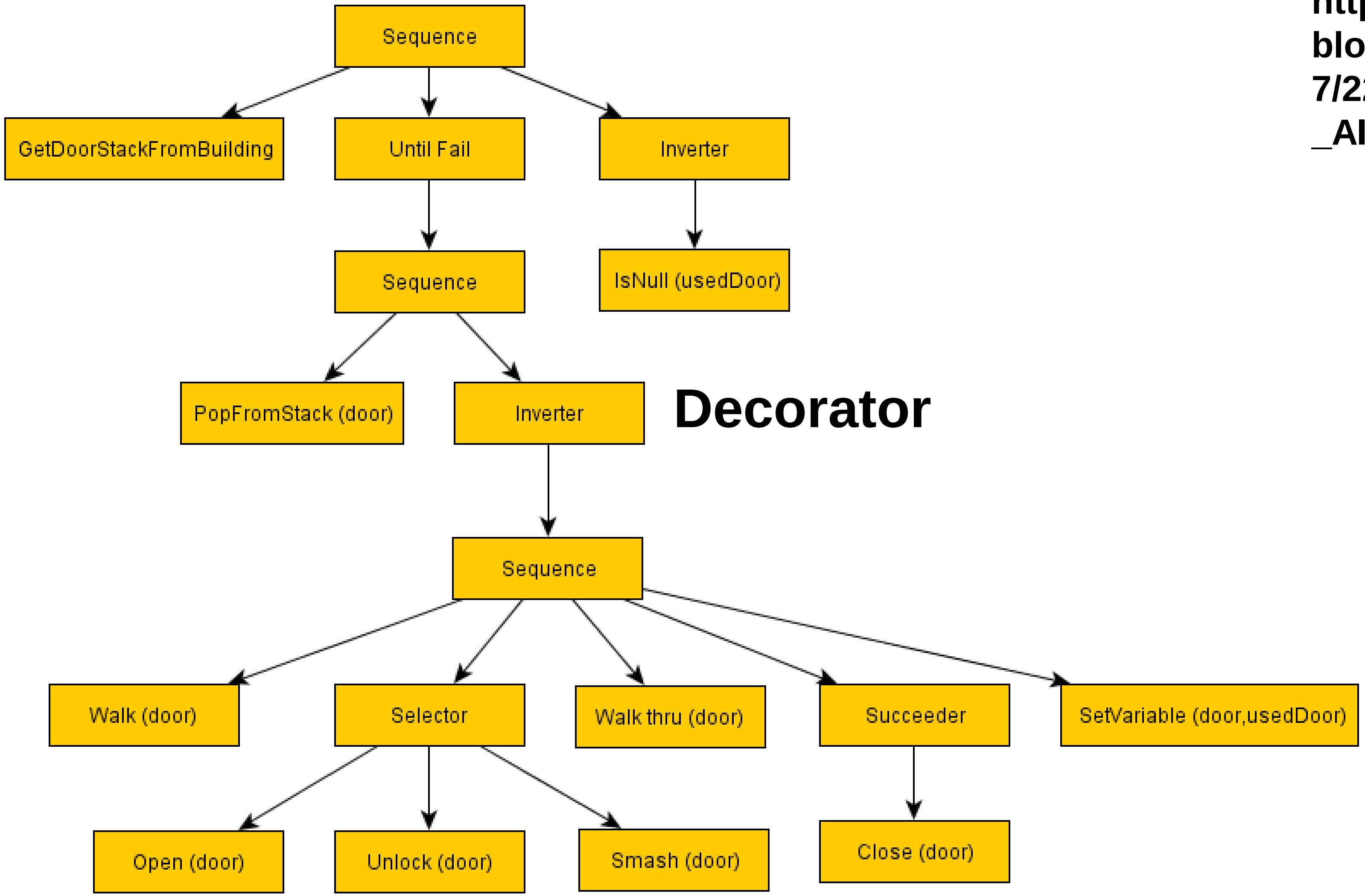
https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

Schematic examples



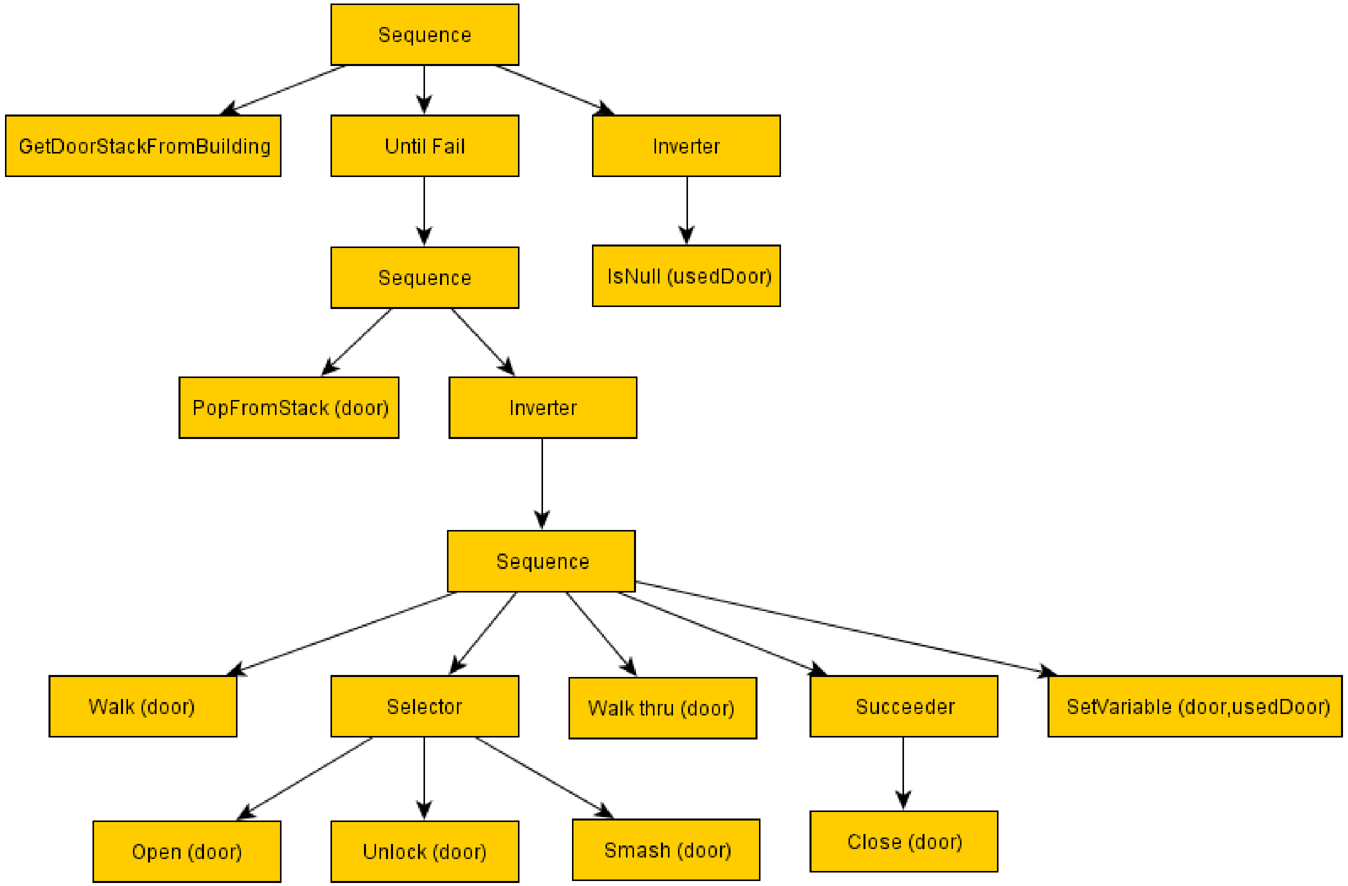
https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

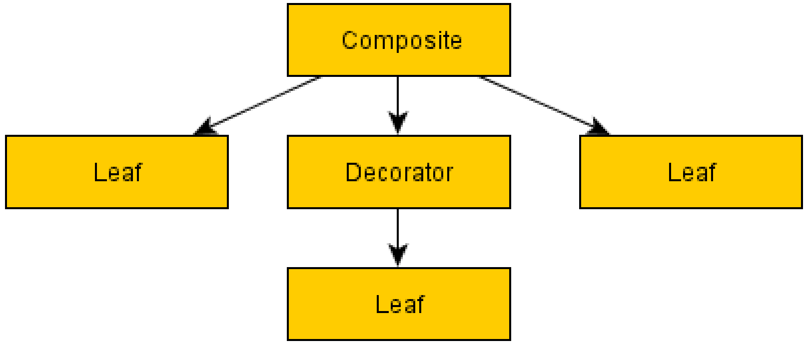
And more complex…



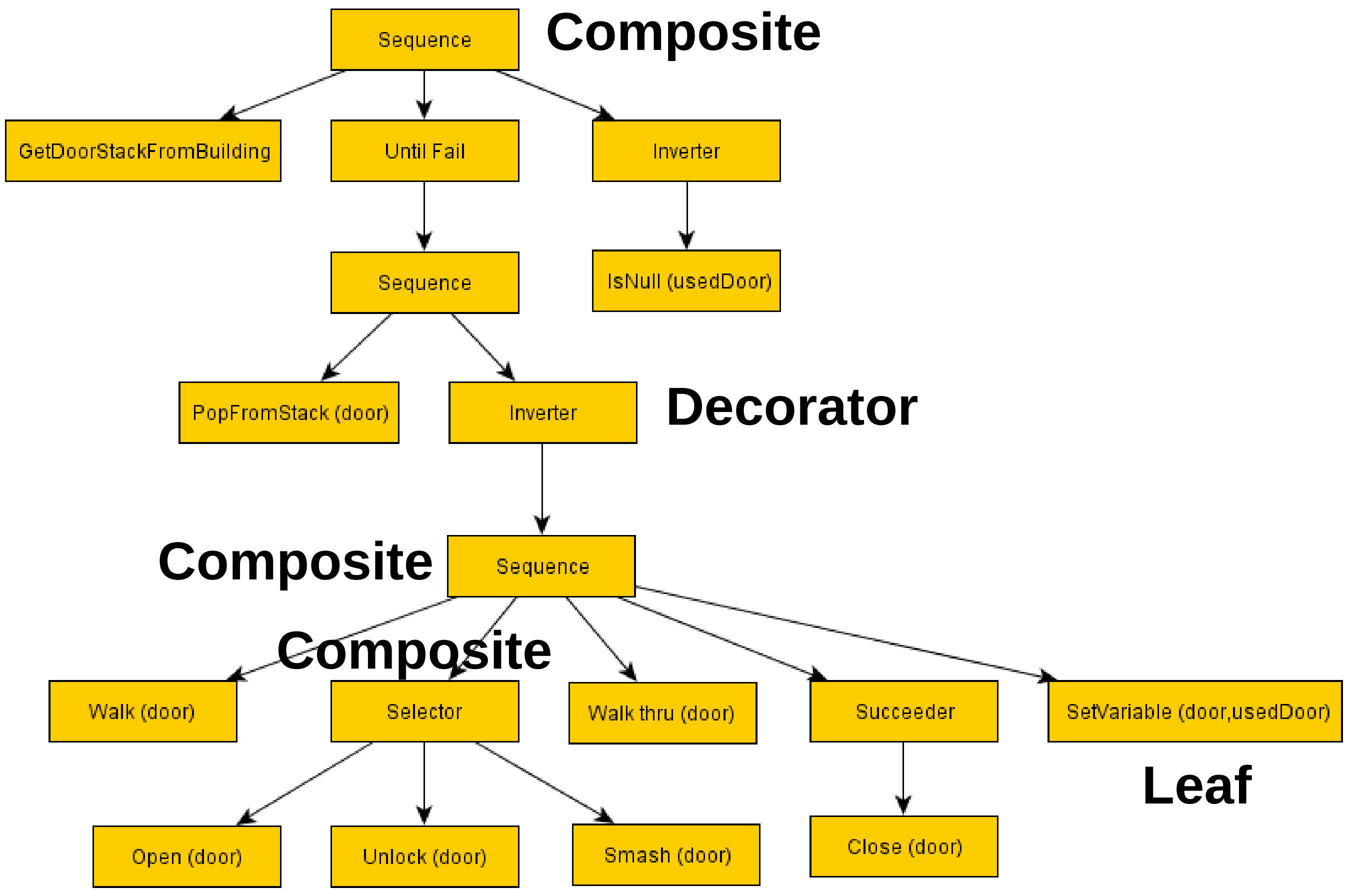
https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

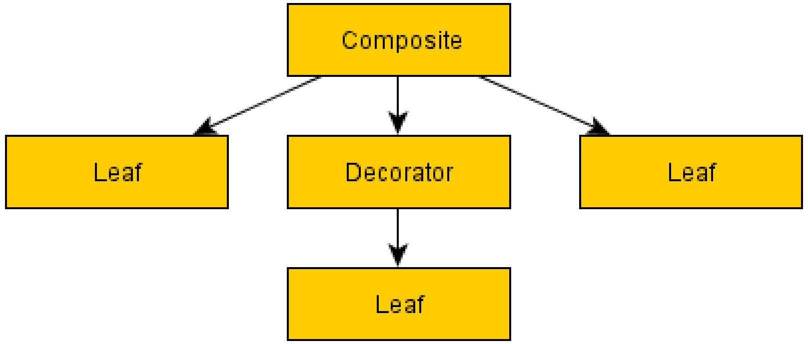
Types


Sequence
GetDoorStackFromBuilding
Until Fail
Inverter
Sequence
IsNull (usedDoor)
PopFromStack (door)
Inverter
Sequence
Walk (door)
Selector
Walk thru (door)
Succeeder
SetVariable (door,usedDoor)
Open (door)
Unlock (door)
Smash (door)
Close (door)


Composite
Leaf
Decorator
Leaf
Leaf

Types


Sequence
Composite
GetDoorStackFromBuilding
Until Fail
Inverter
Sequence
IsNull (usedDoor)
PopFromStack (door)
Inverter
Decorator
Composite
Sequence
Composite
Walk (door)
Selector
Walk thru (door)
Succeeder
SetVariable (door,usedDoor)
Leaf
Open (door)
Unlock (door)
Smash (door)
Close (door)


Composite
Leaf
Decorator
Leaf
Leaf

Behaviour Tree Elements

- leaves, are the actual commands that control the AI entity
  - upon tick, return: Success, Failure, or Running

- branches are utility nodes that control the AI's walk down the tree
  - loop through leaves: first to last or random
  - inverter: turn Failure -> Success
  - to reach the sequences of commands best suited to the situation

- trees can be extremely deep
  - nodes calling sub-trees of reusable functions
  - libraries of behaviours chained together

Analogy

- think of composites and decorators as
- simple functions: negate, …
- if statements, while loops, … for defining flow
- leaf nodes are game specific functions that actually do the work

Examples:

- walk to destination
  - *using shortest path*
  - *success upon reaching the destination*
- avoid salmon, until at distance
- go straight

Behaviour Trees are Modular!

- Can re-use behaviours for different purposes
- Can implement a behaviour as a smaller FSM
- Can be data-driven (loaded from a file, not hard coded)
    - *JSON?!*
- Can easily be constructed by non-programmers
- Can be used for *goal based programming*

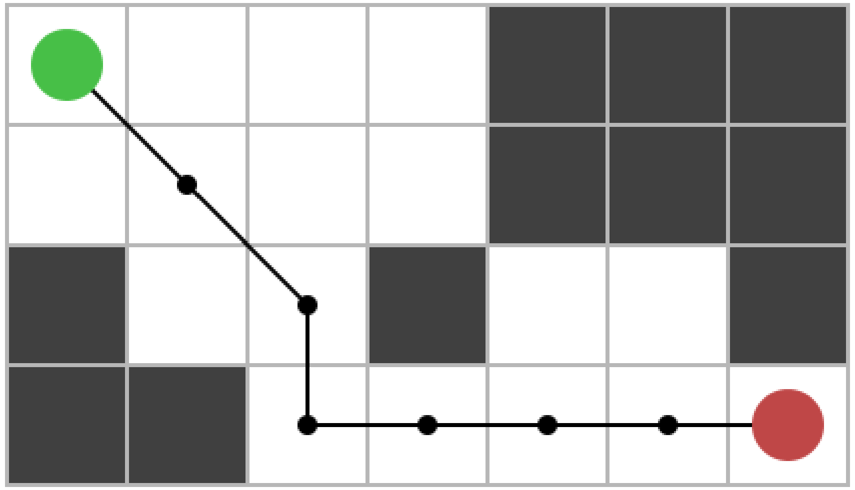
Strategy

Strategy

- Given current state, determine **BEST** next move
- Short term: best among immediate options
- Long term: what brings something closest to a goal
  - *How?*
    - Search for path to best outcome
      - Across states/state parameters

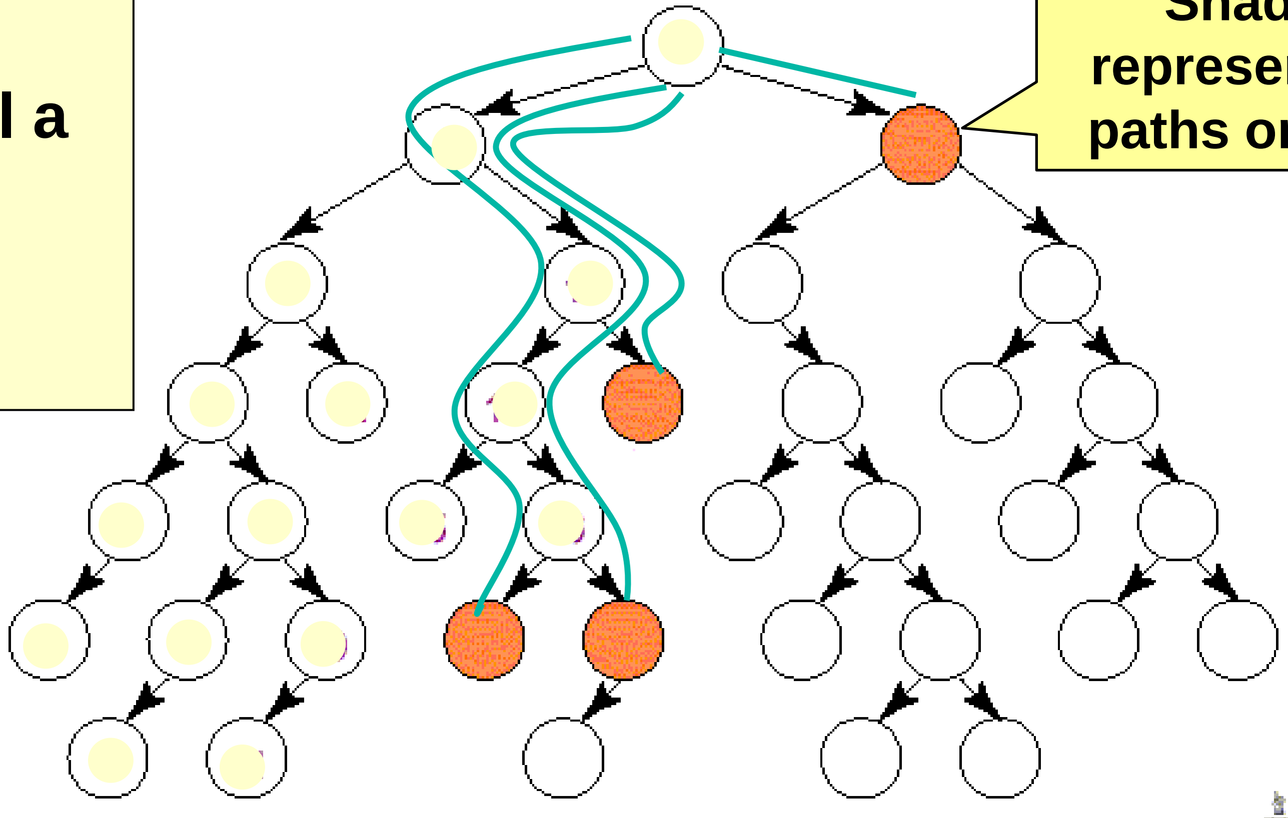
Pathfinding

- **How do I get from point A to point B?**



DFS: Depth First Search

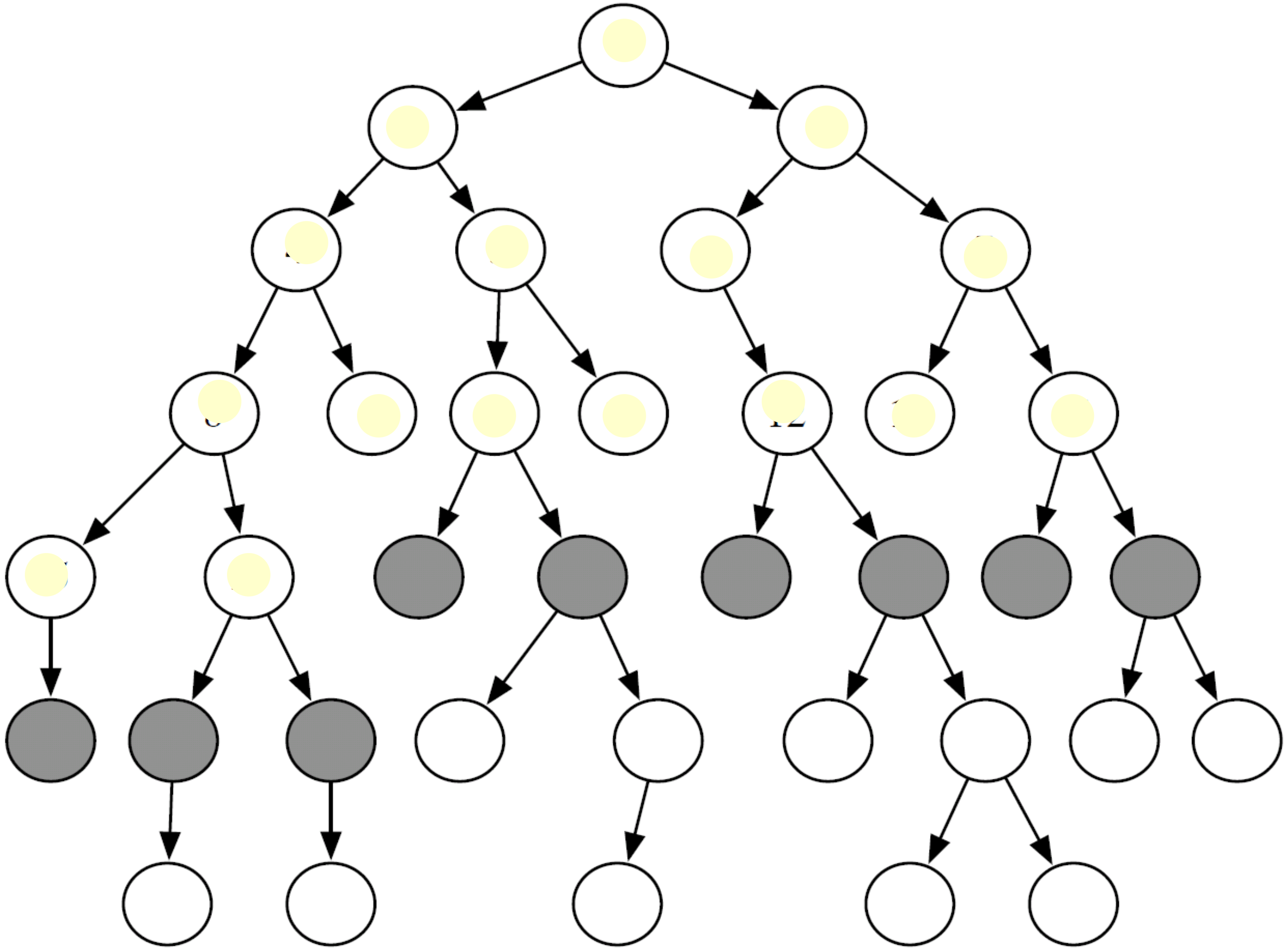
Explore each path on the frontier until its end (or until a goal is found) before considering any other path.



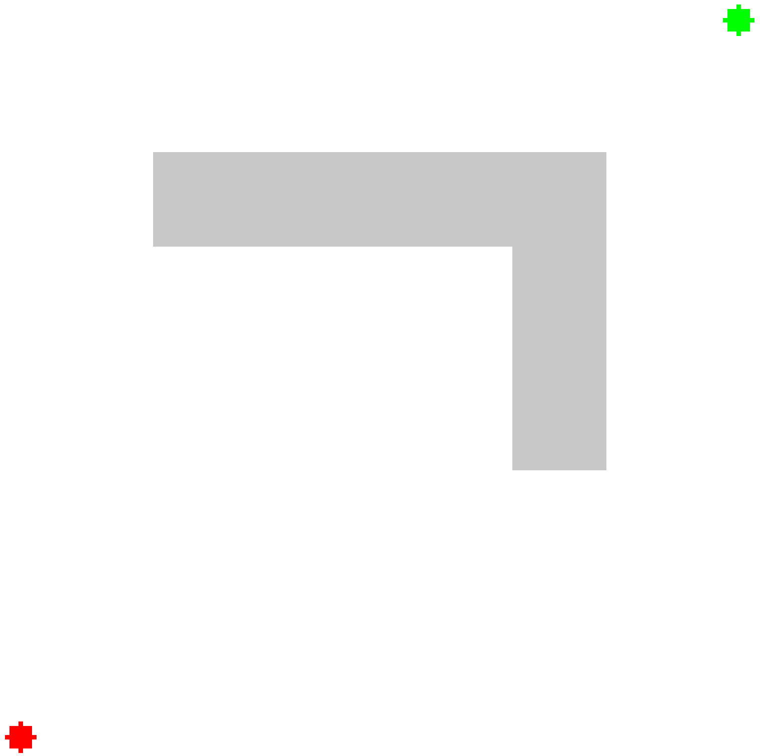
Shaded nodes represent the end of paths on the frontier

Breadth-first search (BFS)

- **Explore all paths of length L on the frontier, before looking at path of length *L + 1***



Breadth-first



https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

When to use BFS vs. DFS?

- ***The search graph has cycles or is infinite***

  BFS

- ***We need the shortest path to a solution***

  BFS

- ***There are only solutions at great depth***

  DFS

- ***There are some solutions at shallow depth***

  BFS

- ***No way the search graph will fit into memory***

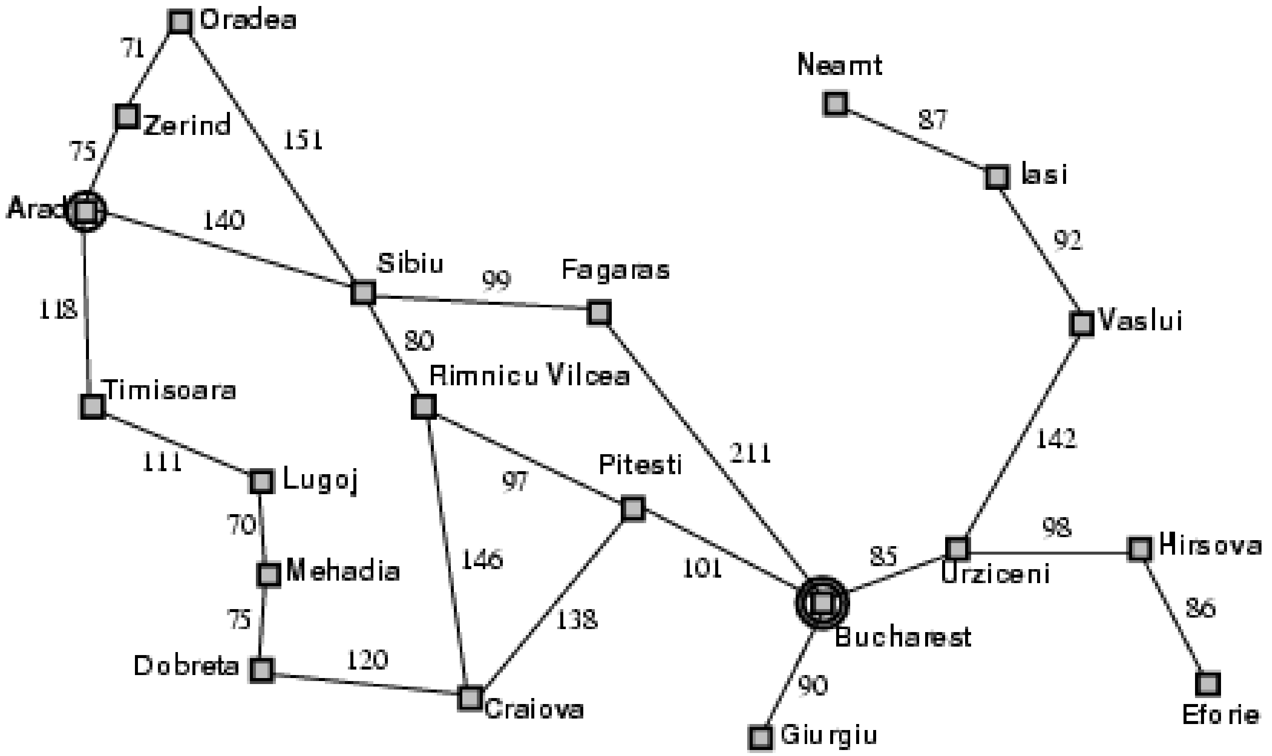
  DFS

Search with Costs

Def.: The cost of a path is the sum of the costs of its arcs

$$\text{cost}(\langle n_0, \ldots, n_k \rangle) = \sum_{i=1}^{k} \text{cost}(\langle n_{i-1}, n_i \rangle)$$

Want to find the solution that minimizes cost



Lowest-Cost-First Search (LCFS)

- Lowest-cost-first search finds the path with the lowest cost to a goal node
- At each stage, it selects the path with the lowest cost on the frontier.
- The frontier is implemented as a priority queue ordered by path cost.

Use of search

- Use search to determine next state (next state on shortest path to goal/best outcome)
- Measures:
    - *Evaluate goal/best outcome*
    - *Evaluate distance (shortest path in what metric?)*

Problems:

- Cost of full search (at every step) can be prohibitive
- Search in adversarial environment
    - *Player will try to outsmart you*

Heuristic Search

- Blind search algorithms do not take goal into account until they reach it

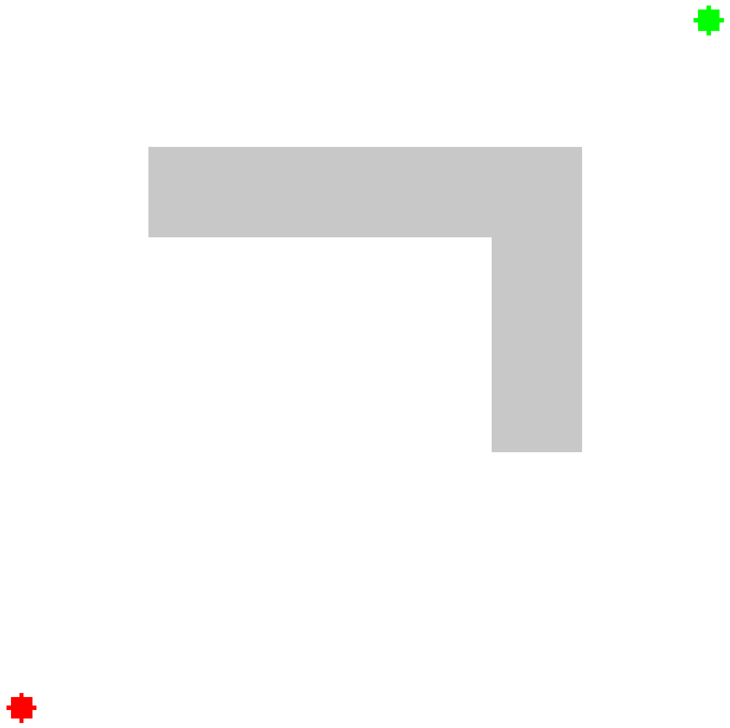
- We often have estimates of distance/cost from node n to a goal node

- **Estimate = search heuristic**
  - **a scoring function h(x)**

Best First Search (BestFS)

- Best First: always choose the path on the frontier with the smallest h value
  - *Frontier = priority queue ordered by h*
  - *Once reach goal can discard most unexplored paths…*
    - Why?
  - *Worst case: still explore all/most space*
  - *Best case: very efficient*
- **Greedy**: (only) expand path whose last node seems closest to the goal
  - *Get solution that is* ***locally*** *best*

A* search



https://en.wikipedia.org/wiki/A*_search_algorithm