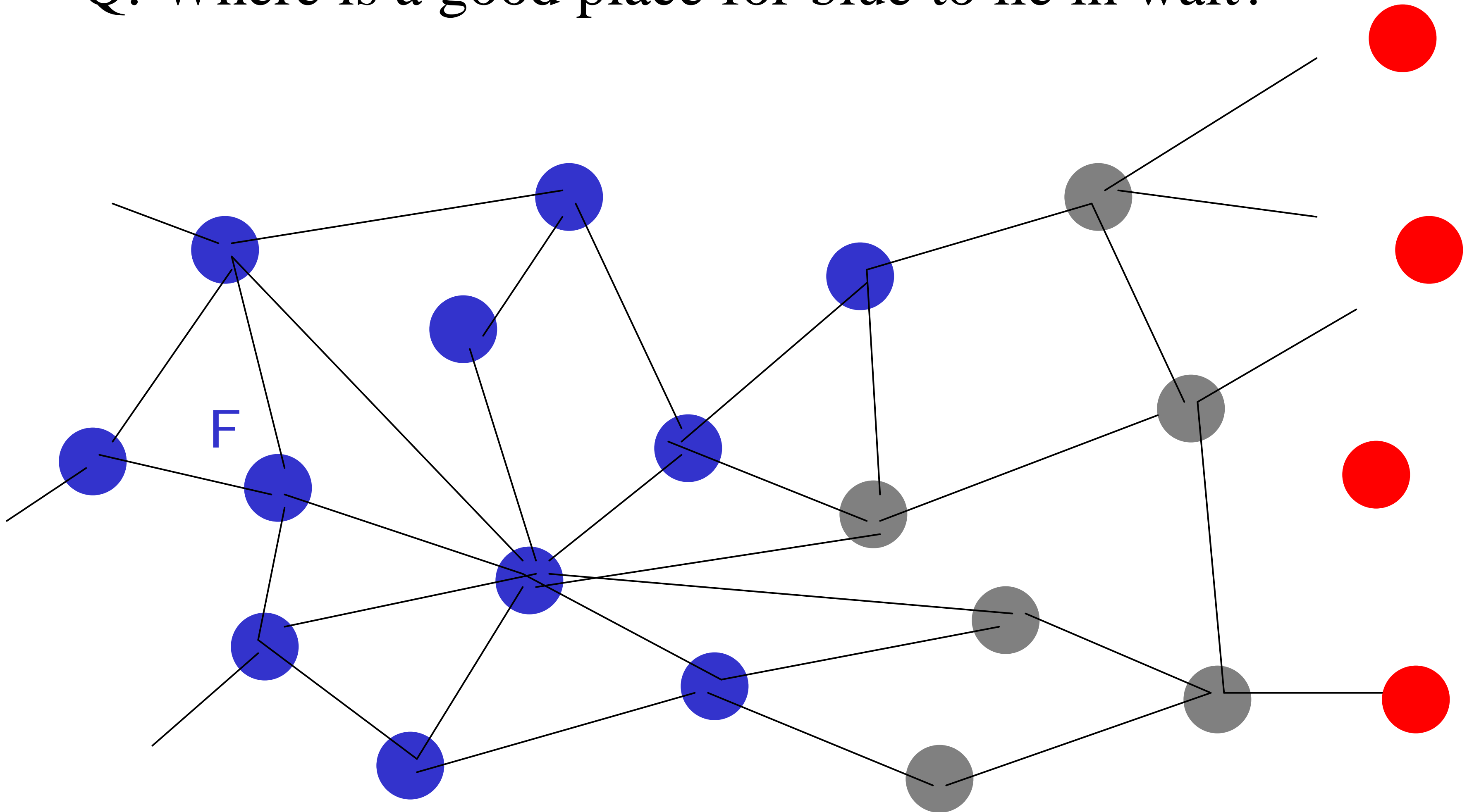
Capture the Flag

& Related Topics

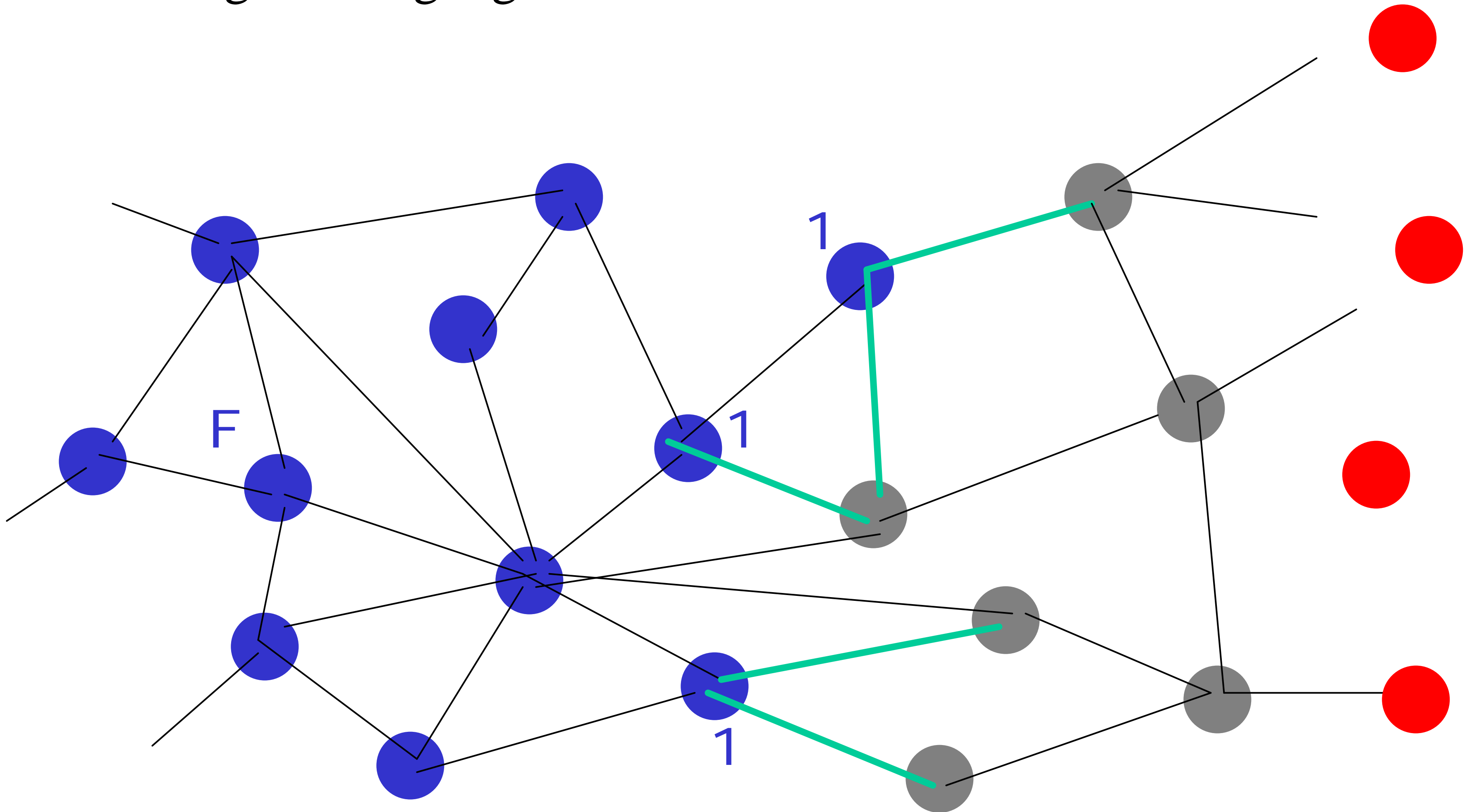
Strategies?

- Offensive strategies, defensive strategies, others?
- Q: Where is a good place for blue to lie in wait?



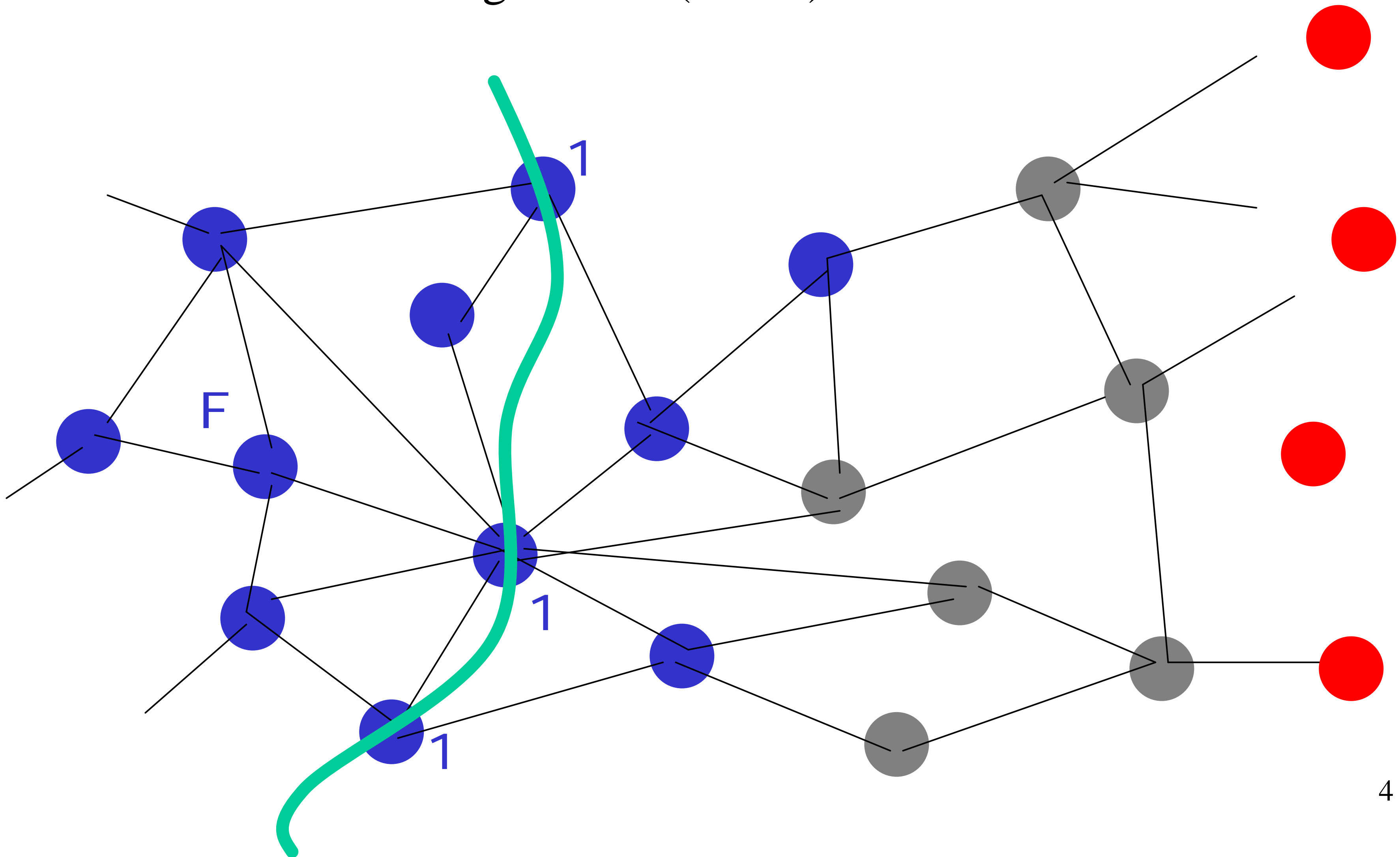
Patrol the border

- Wait in blue zone just on edge of neutral areas
  - → Edge finding algorithms



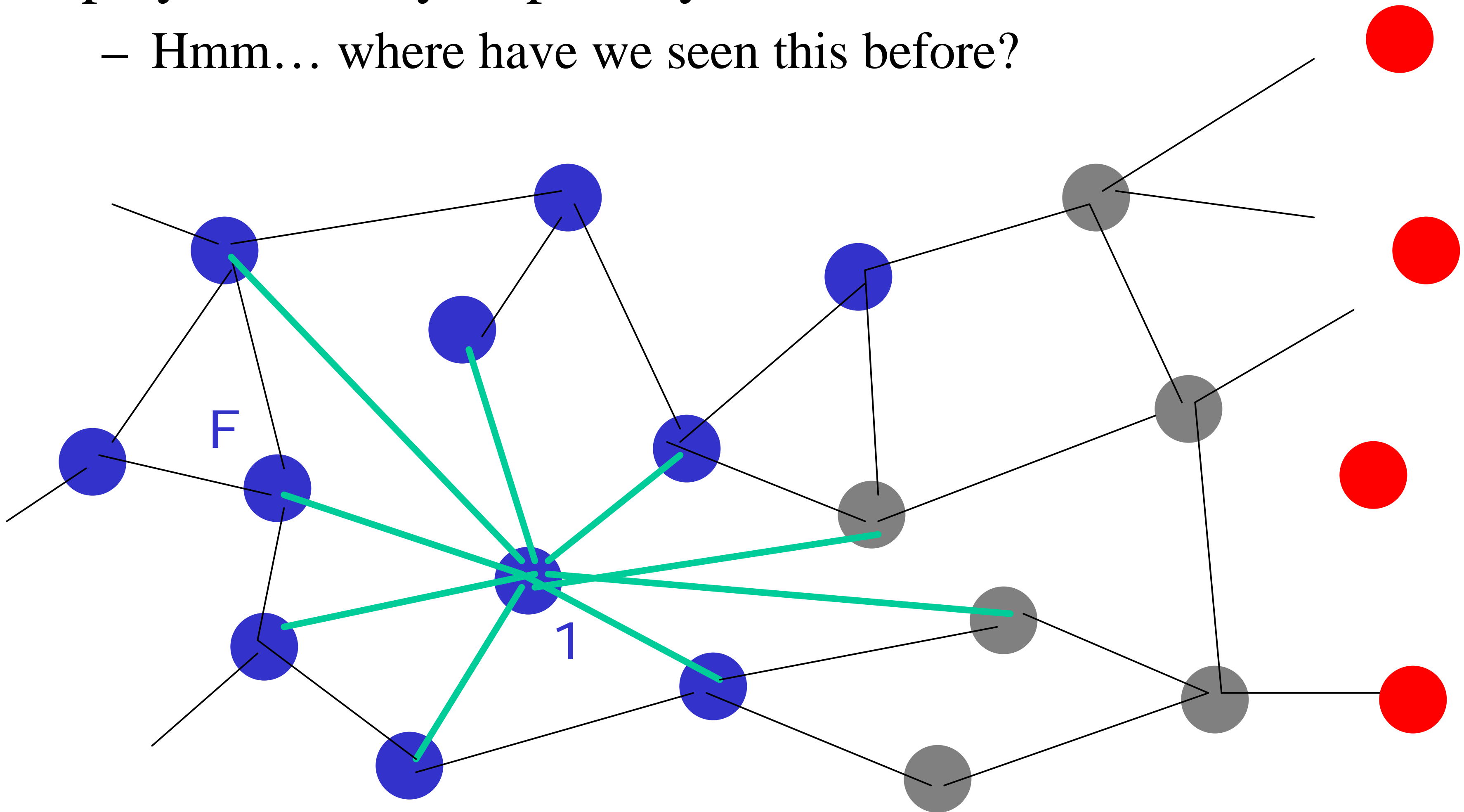
"Cut" the graph

- Attempt to isolate the flag with a line of blue players
  - → Minimum cut algorithms (cs4xx)



Wait at “important” nodes

- Find nodes that are on many paths, where a random player is likely to pass by
  - Hmm… where have we seen this before?



Shortest Path Algorithms

- In our game, all edges have weight = 1
  - Yes, a BFS traversal will find the minimum path
- Basic algorithm (from lecture), using a Queue for toDo set

```
Add root node to toDo
shortestpath[root] = emptypath
Repeat:
    u = take next node out of toDo
    For each neighboring node v
        If shortestpath[v] not yet computed
            path = same as shortestpath(u), with new edge u-v stuck on end
            shortestpath[v] = path
            add u to toDo set
```

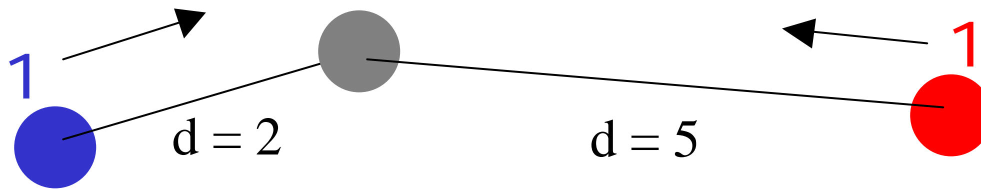
- Watch out for cycles in graph!
- How to store path? How to store shortestpath table?

Why didn't we use edge weights?

- weight = cartesian distance
  = sqrt((x1-x2)^2+(y1-y2)^2)
- Hint: It's not because we don't like Dijkstra

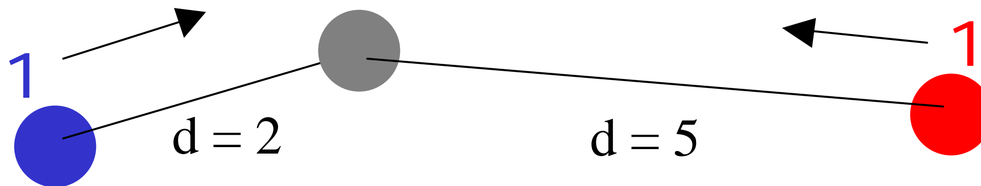
Progression of game

- At start of game, suppose both teams go somewhere
- What happens?

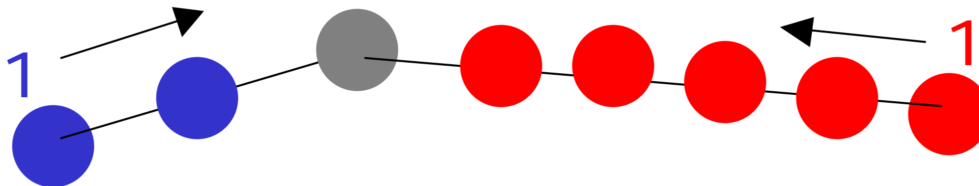


Use penalty rounds

- Blue sits out for 2 turns, Red sits out for 5 turns

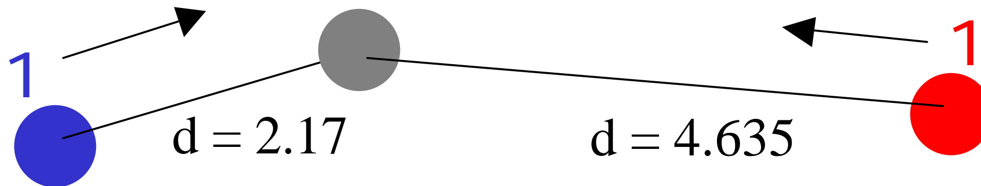


- Why not just add more nodes, using weight = 1?



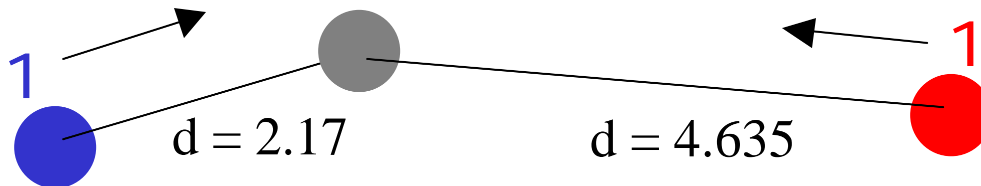
Fractional rounds

- What to do if distances not whole numbers?



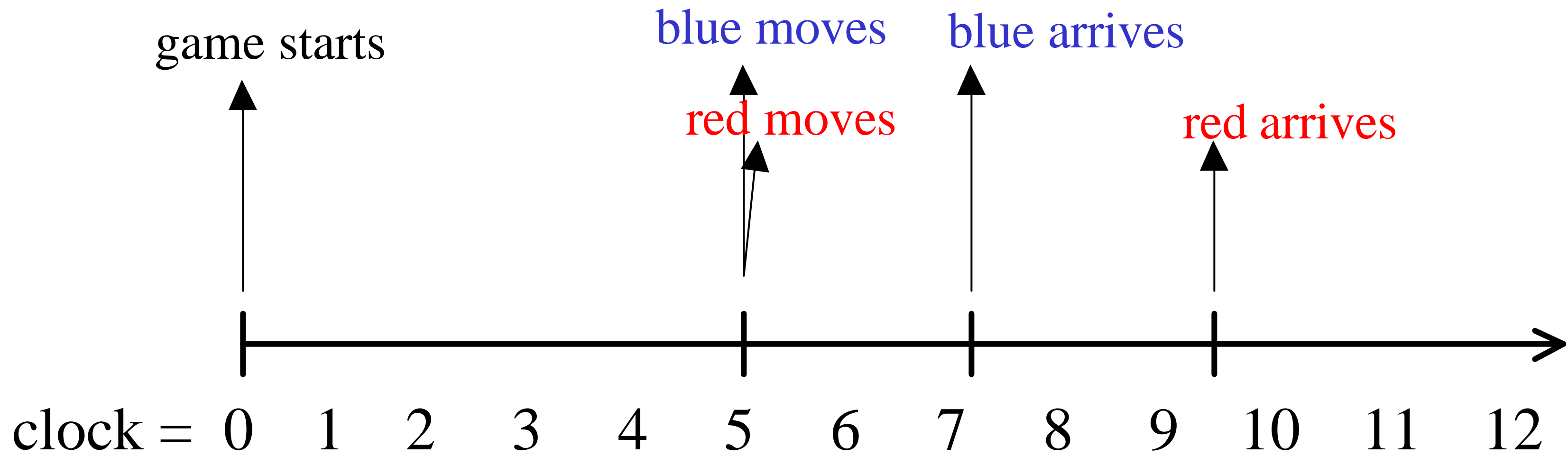
Fractional rounds

- What to do if distances not whole numbers?



- Keep a "clock" in the game
  - If blue moves at time 5, she arrives at time 7.17
  - If red moves at time 5, he arrives at time 9.635
- Player gets to move again when clock is equal to the arrival time

Timeline



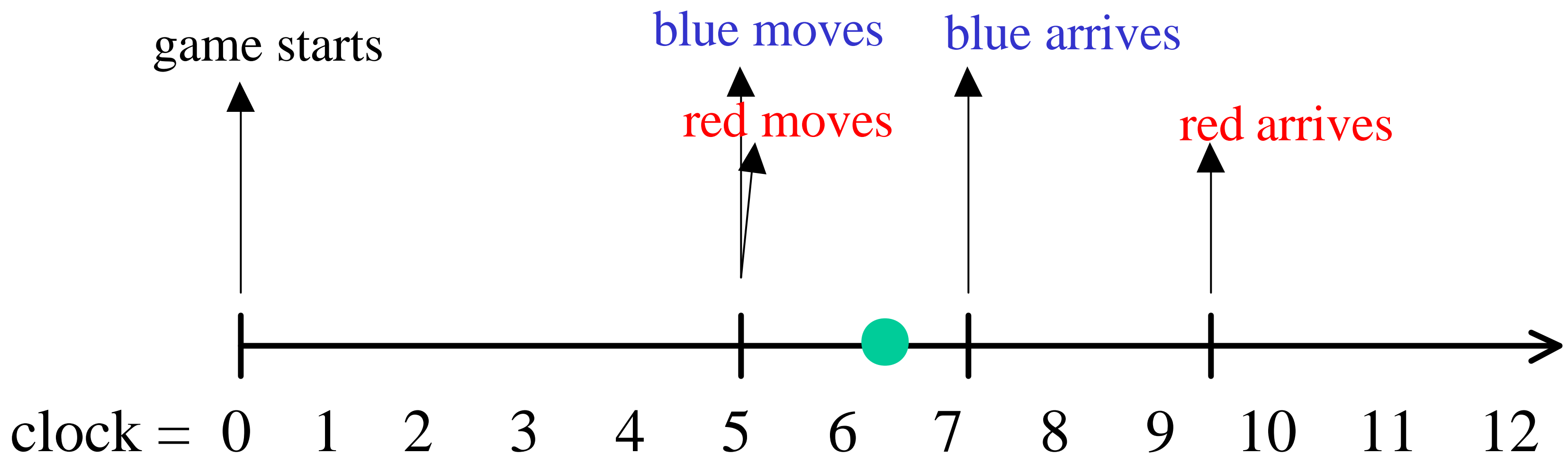
Basic Game Loop

- Was:

```
for (round = 0; round < infinity; round++)
    let each player move once
```

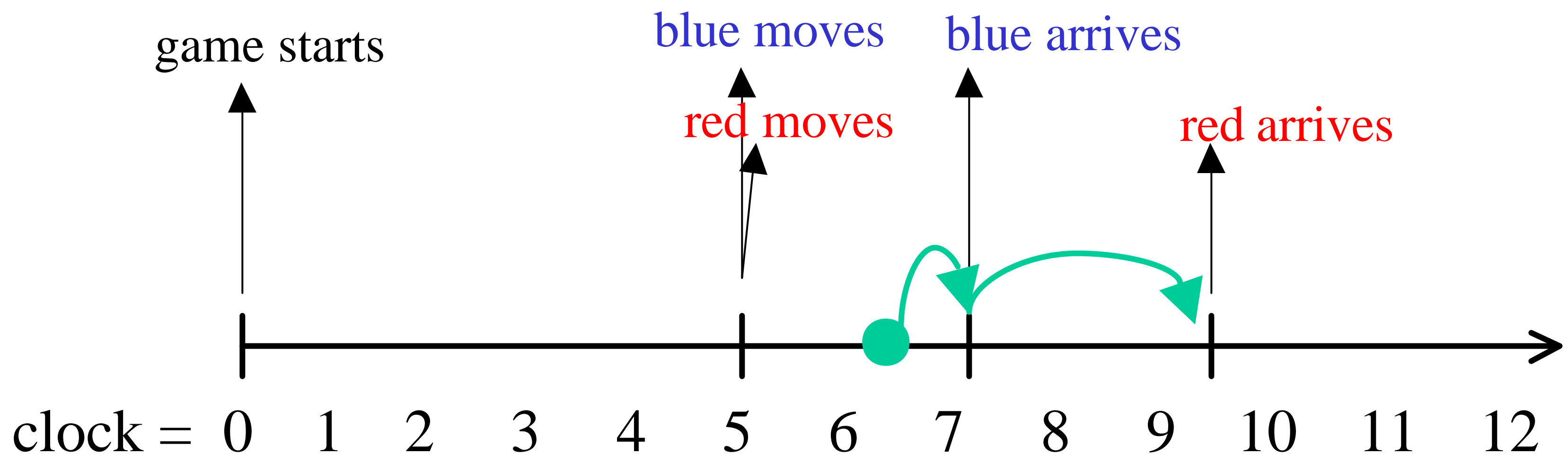
- Now:

```
while (t < infinity)
    if there are things to do at time t
        do them
    increment the clock value t
```



Updating the clock

- If nothing to do at time t, advance clock to next thing that will happen
  - e.g. t = 6.3; next thing is at 7.17, so advance to t = 7.17
  - e.g. t = 7.17; next thing is at 9.635…



Discrete Event Simulation

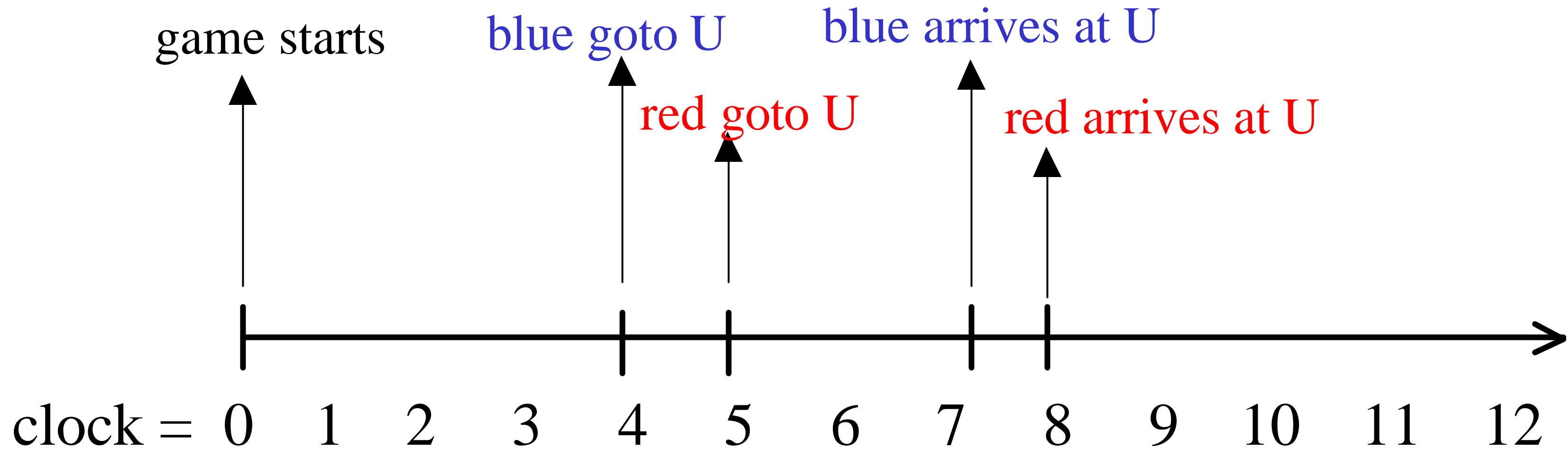
- Consists of:
  - *entities*
    - schedule events when they want to do something
  - *events*
    - are scheduled for some time t
    - fire when “clock” reaches time t
    - do something when fired
  - *scheduler*
    - maintains set of scheduled events
    - maintains a “clock”
    - responsible for deciding which event fires next

Entities & Events

- In Capture the Flag…
- Entities are players
  - They do things directly (e.g., compute shortest path trees, decide what move to do, change their state)
  - But also schedule *events* to happen in the "future"
- Events are things that happen to players at a specific time in the future
  - Not the same thing as actions!
- E.g. schedule ArriveAtPlace event for time $t+2.17$
  - When event fires, player gets to make another decision
    - Which will schedule more future events…

Much harder than "rounds"

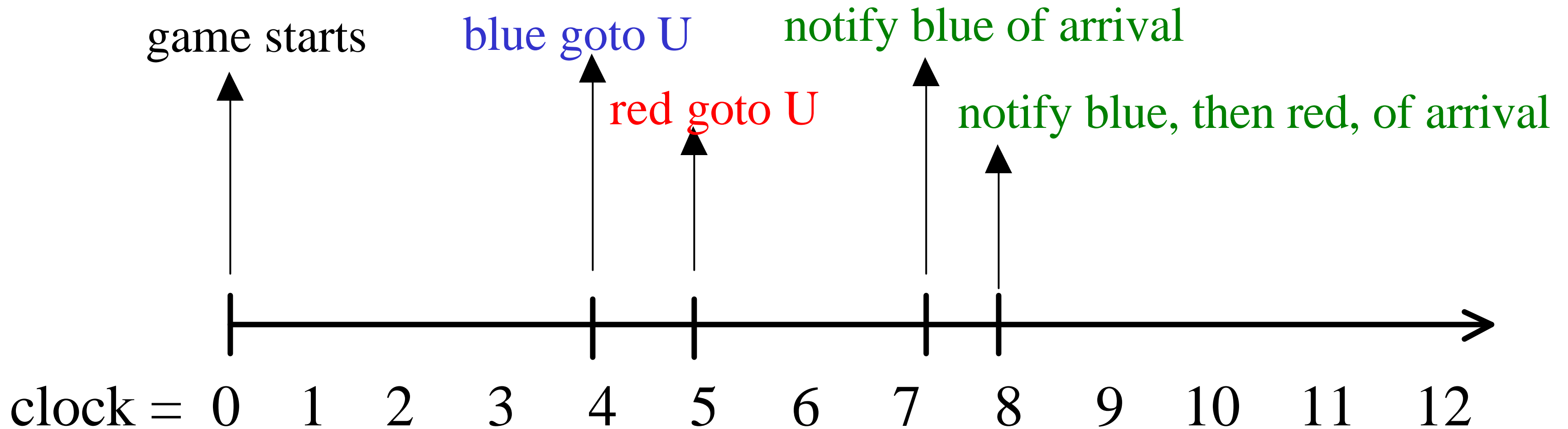
- Entity does not "see" other events synchronously



- Who is going to have the advantage here?

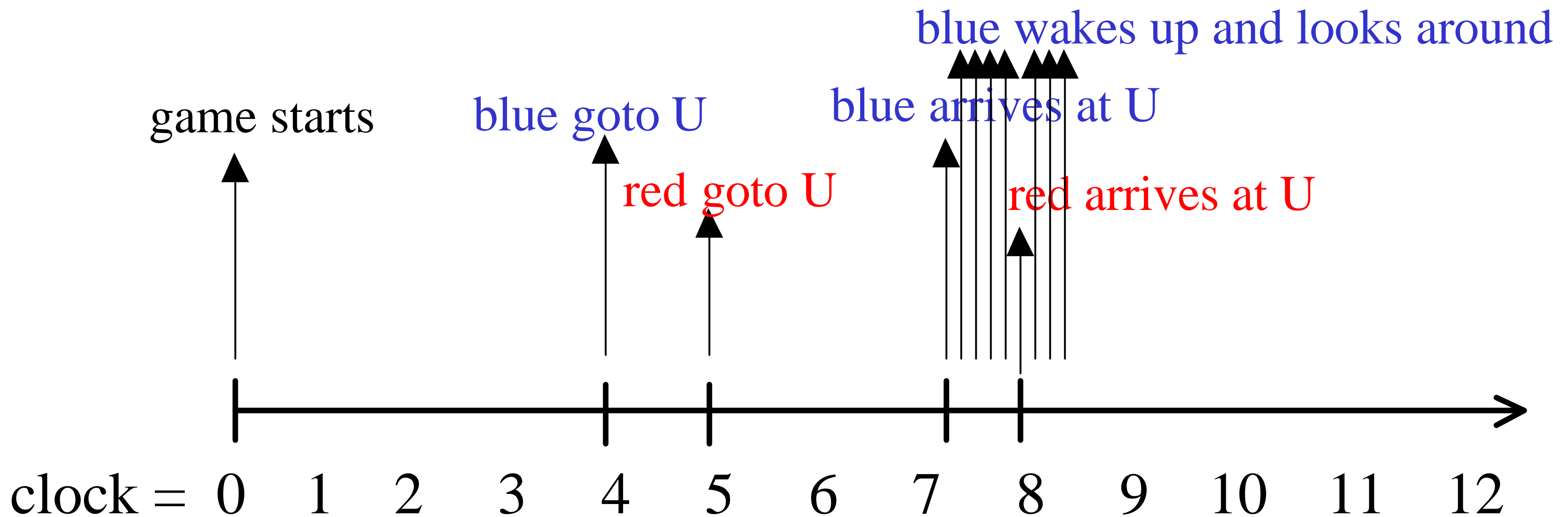
Approaches

- The "ArriveAtPlace" event could notify all the entities in the room
  - Perhaps in order of which entity was there the earliest?
  - Or random order?



Approaches

- Could require entities to pay closer attention
- Schedule a "WakeUpAndLookAround" event for time $t+0.001$



Scheduler

- scheduler.schedule(event *e* at time *e*.*t*)
- scheduler.cancel(event *e* at time *e*.*t*)
- scheduler.infiniteLoop():

```
t = 0.0;
while (t < infinity) {
        e = event scheduled for earliest time
        if (e.t == t)
                e.fire();
        else
                t = e.t
}
```

Scheduler data structures

- Scheduler maintains an "event queue"
  - Not really a FIFO "queue", because events are scheduled in any order, but are processed lowest-time-first-out
- scheduler.schedule(event e at time $e.t$)
  - insert e into event queue at correct place
- scheduler.cancel(event e at time $e.t$)
  - find and remove e from event queue
- scheduler.infiniteLoop():
  - finds and removes event with smallest time

Priority Queue

- Called a Heap-Scheduler
- Maintain heap ordering using event times
  - Implement *compareTo* by comparing *lhs.t* and *rhs.t*
- schedule() – O(log n) // n events in queue
- remove_min() – O(log n) // n events in queue
- cancel() – O(n) // ouch!

- Used commonly in practice
  - cancel() turns out to be rare for many simulations
  - But: most events are scheduled for near future
    - consequences?

Sorted List

- Called a List-Scheduler
- Maintain list ordering using event times
  - Implement *compareTo* by comparing *lhs.t* and *rhs.t*
- schedule() – O(n) // ouch!
- remove_min() – O(1) // linked list
- cancel() – O(n) // ouch!

- Used commonly in practice
  - cancel() turns out to be rare for many simulations
  - And: most events are scheduled for near future
    - consequences?

Why not Discrete Event Simulation?

- For capture the flag?
  - Usually very tricky to program a DES
  - No threads at all – everything is “discrete events”
  - Intuitive notion of time not obeyed
    - Computation happens in zero simulated time
    - So can’t have a while-loop looking for some condition to become true

Threads: An Alternative

- Idea:
- Every Player gets a thread
- Moves as often as they wish or can
- Pros?

Threads: An Alternative

- Idea:
- Every Player gets a thread
- Moves as often as they wish or can
- Pros?
  - Can use edge weights/distances
    - thread goes to sleep for specified time
  - Computational efficiency really matters
    - faster implementations get to make more moves
  - Players act independently, as they would in real life

Cons

- Too many threads degrades Java performance
  - Someday this will be fixed, maybe
  - Same problem (to varying extents) with many languages
- Thread synchronization is a hard problem
  - Even for the best programmers, mistakes are common
  - See Mars Rover incident
- Too open to evil implementations
  - Kill/stall/interrupt other threads
  - synchronize on the game object, no one else can do anything
  - raise your own priority
  - Do lots of disk I/O to interfere with other threads
  - etc.