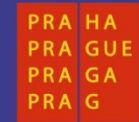




OPERAČNÍ PROGRAM PRAHA  
ADAPTABILITA



EVROPSKÁ UNIE

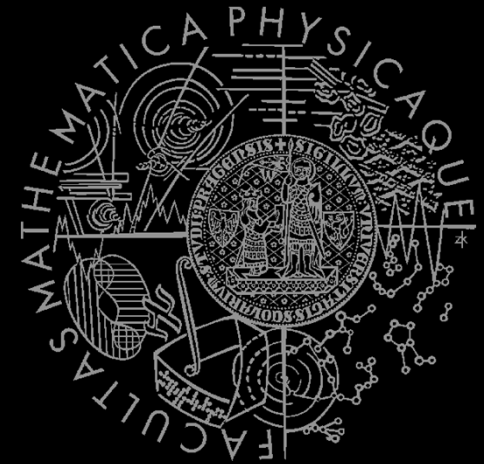
EVROPSKÝ SOCIÁLNÍ FOND

# Pogamut 3

## Lekce 6 – A\* + Viditelnost

**PRAHA & EU**  
INVESTUJEME DO VAŠÍ BUDOUCNOSTI

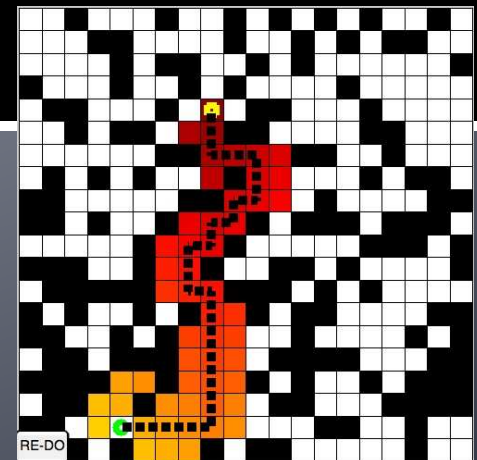
Faculty of mathematics and physics  
Charles University in Prague  
4<sup>th</sup> April 2013



UT2004 bots made easy!

# Pogamut 3

Lecture 6 – A\* + Visibility



# Assignment 5 Revisited

## NavigationBot



- How to detect that the bot has stuck?
- What if the location is currently unreachable?
  - TabooSet explained

# Tag! Tournament

## Debriefing



- *To be found in different presentation...*

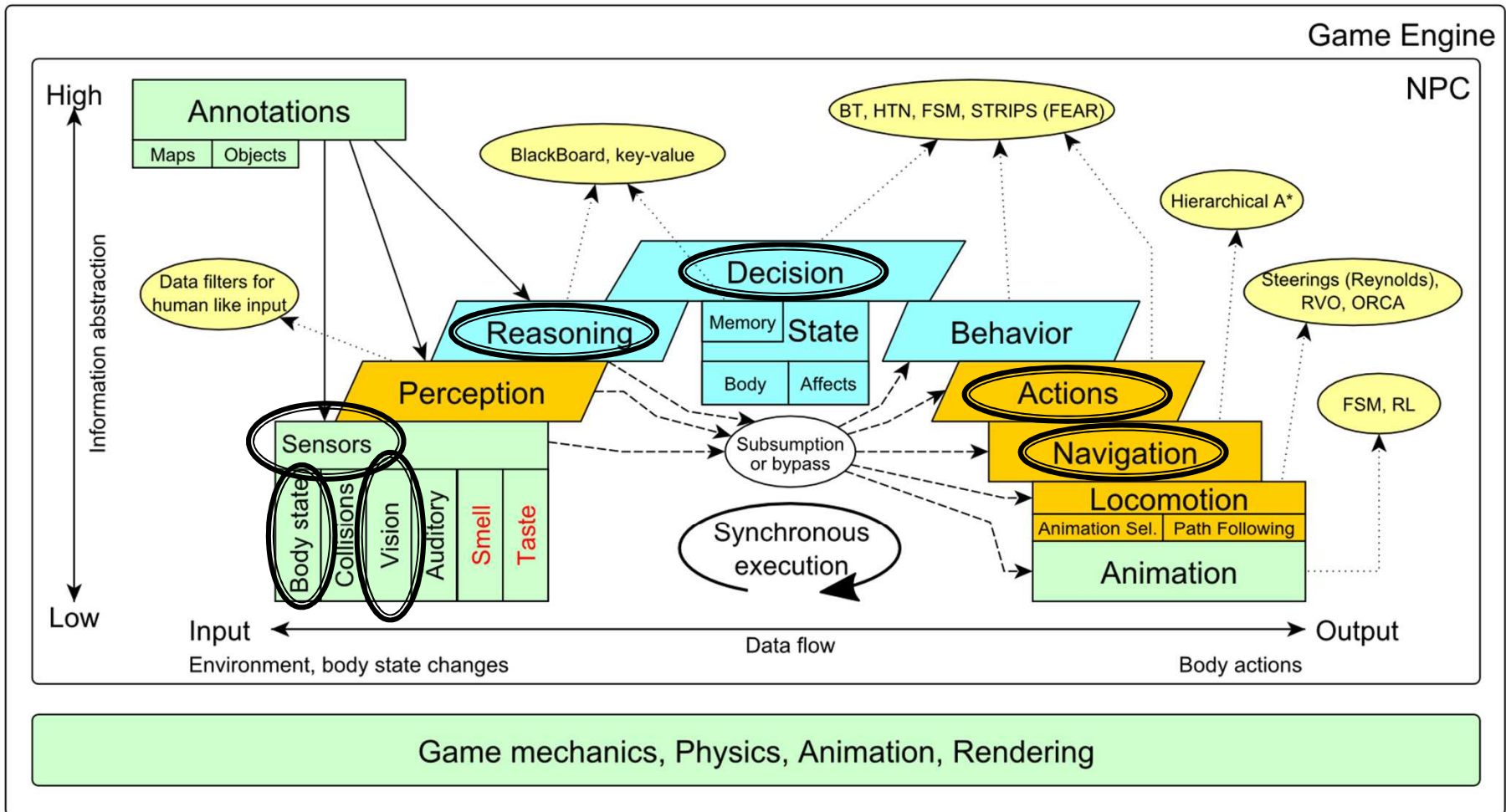
# Today's menu



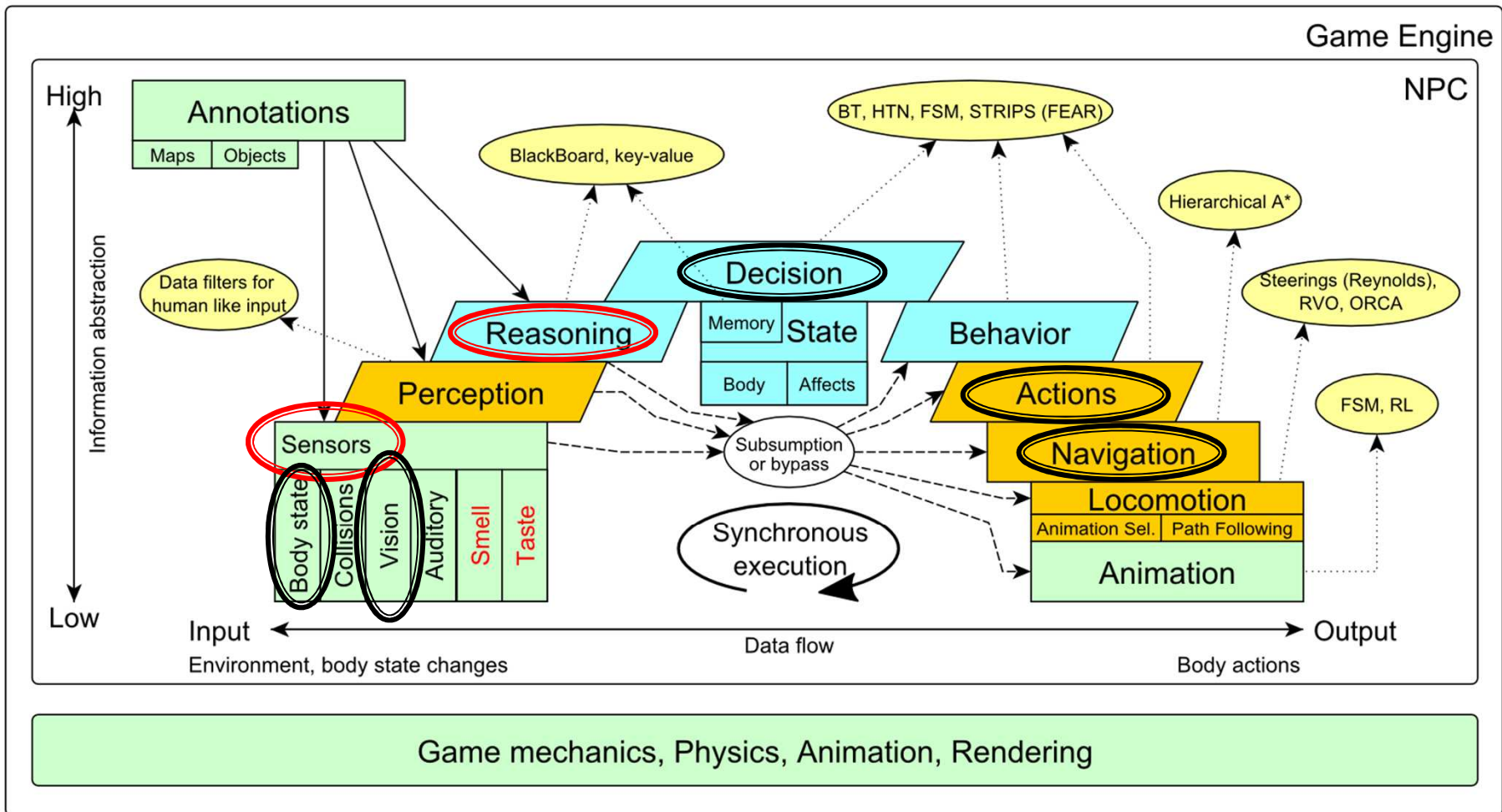
1. **Big Picture**
2. Visibility abstraction
  - Visibility matrix
  - Visibility
  - `this.visibility`
3. How to reason about path
  - A\* and custom map view
  - `UT2004AStar, IPFMapView<NavPoint>`
  - `this.aStar`
4. Hide&Seek Game
  - Rules, Map
  - `HideAndSeekMap`
5. Hide&Seek Tournament Announcement

# Big Picture

## Already covered



# Big Picture Today



# Today's menu



1. Big Picture
2. **Visibility abstraction**
  - Visibility matrix
  - Visibility
  - `this.visibility`
3. How to reason about path
  - A\* and custom map view
  - `UT2004AStar`, `IPFMapView<NavPoint>`
  - `this.aStar`
4. Hide&Seek Game
  - Rules, Map
  - `HideAndSeekMap`
5. Hide&Seek Tournament Announcement

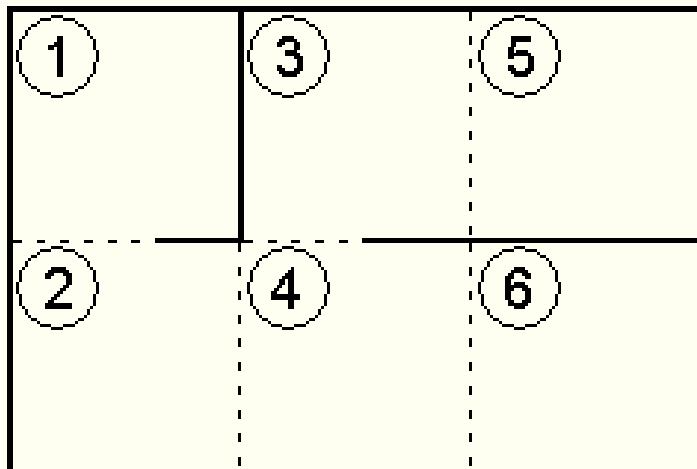


# Visibility Abstraction

## Visibility Matrix



- visibility class
  - Contains precomputed visibility matrix between path points and some points on links
  - Matrices for competition maps already present



	1	2	3	4	5	6
$V_1$	1	1	0	0	0	0
$V_2$	1	1	0	1	0	1
$V_3$	0	0	1	0	1	0
$V_4$	0	1	0	1	0	1
$V_5$	0	0	1	0	1	0
$V_6$	0	1	0	1	0	1

# Visibility Matrix

How to get to cover?



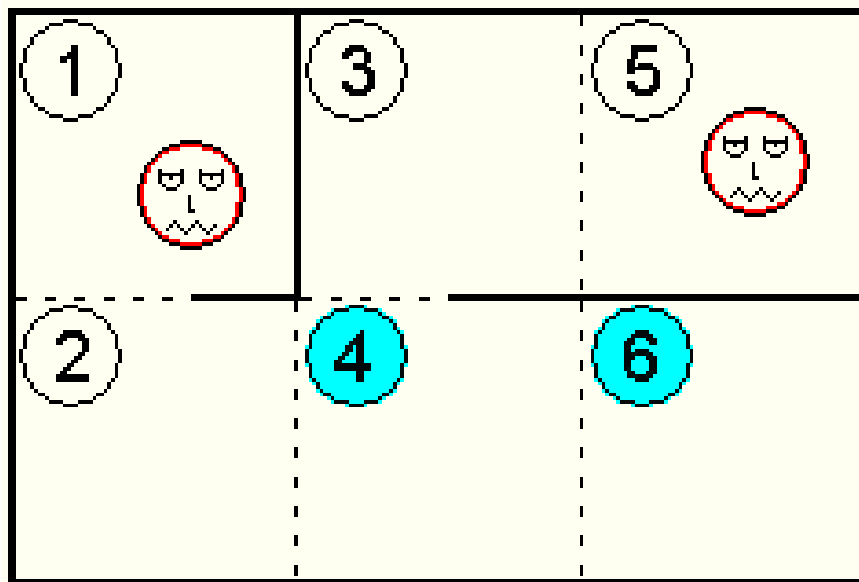
- How to find the cover?

- Enemies ...

$$E_1..E_k$$

- Safe waypoints ...

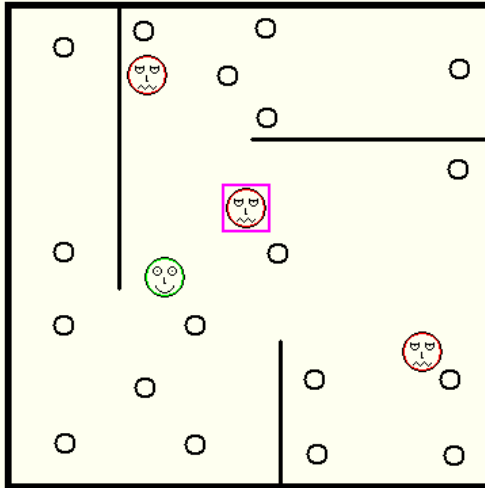
$$S = \neg \bigvee_{i=1}^k V_{E_i}$$



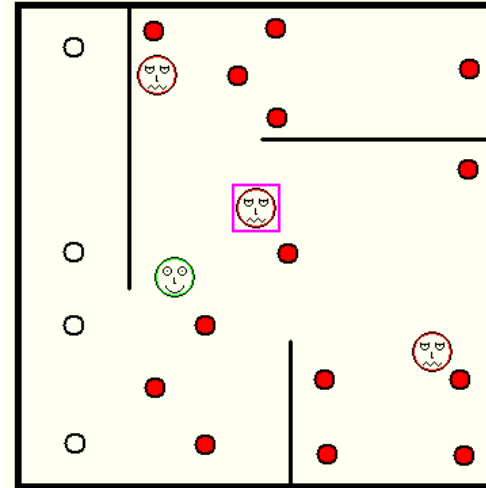
	1	2	3	4	5	6
$V_1$	1	1	0	0	0	0
$V_2$	1	1	0	1	0	1
$V_3$	0	0	1	0	1	0
$V_4$	0	1	0	1	0	1
$V_5$	0	0	1	0	1	0
$V_6$	0	1	0	1	0	1

# Visibility Matrix

## Smart attack

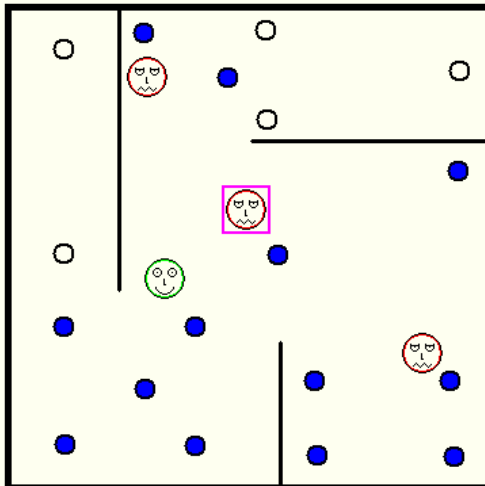


1. Choose target T
2. Others are enemies  $E_i$



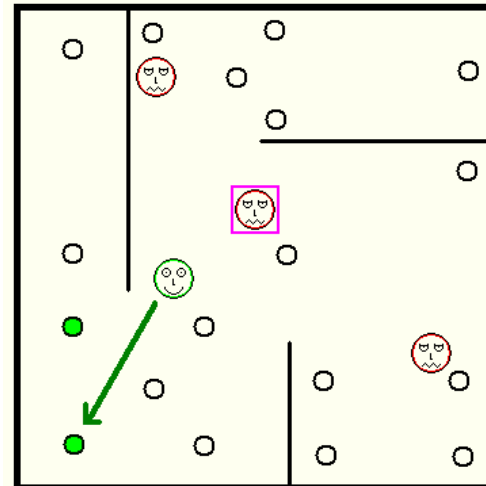
3. Navpoints other enemies  $E_i$  can see

$$\bigvee_{i=1}^k V_{E_i}$$



2. Navpoints target T is visible from

$$V_T$$



4. Smart place to shoot from

$$V_t \wedge \neg \bigvee_{i=1}^k V_{E_i}$$

# Visibility Matrix

## Interesting methods



- `Visibility` class

  - `getNearestVisibilityLocationTo(ILocated)`

  - `getCoverPointsFrom(ILocated)`

  - `getCoverPointsFromN(ILocated...)`

  - `getMatrix()`

- `VisibilityMatrix` class

  - `getMatrix()`

  - `getNearestIndex(ILocated located)`

# Visibility Matrix

## Visibility matrix file



- To be able to use the visibility matrix, you need to have a file with the visibility information
- Each map has its own file. E.g.

`visibilityMatrix-DM-TrainingDay-all.bin`

- Place this file in the root of the project folder of your bot
- Get all matrices from svn

`svn://artemis.ms.mff.cuni.cz/pogamut/trunk/project/  
Main/PogamutUT2004Examples/19-  
VisibilityBatchCreator/visibility-matrices`

# Today's menu



1. Big Picture
2. Visibility abstraction
  - Visibility matrix
  - Visibility
  - `this.visibility`
3. **How to reason about path**
  - A\* and custom map view
  - `UT2004AStar`, `IPFMapView<NavPoint>`
  - `this.aStar`
4. Hide&Seek Game
  - Rules, Map
  - `HideAndSeekMap`
5. Hide&Seek Tournament Announcement

# A\* Algorithm

## Reasoning



- Agent deliberation cycle
  1. Update senses
    - Some Players have become visible
  2. Update percepts
    - They are all enemies!
  3. Reason
    - **Where can I take cover? How can I fallback?**
    - => Infer new information given the senses / percepts**
  4. Decide
    - Inform my team then ... should I take cover, fallback or attack?
  5. Take action

# A\* Algorithm

## Dijkstra



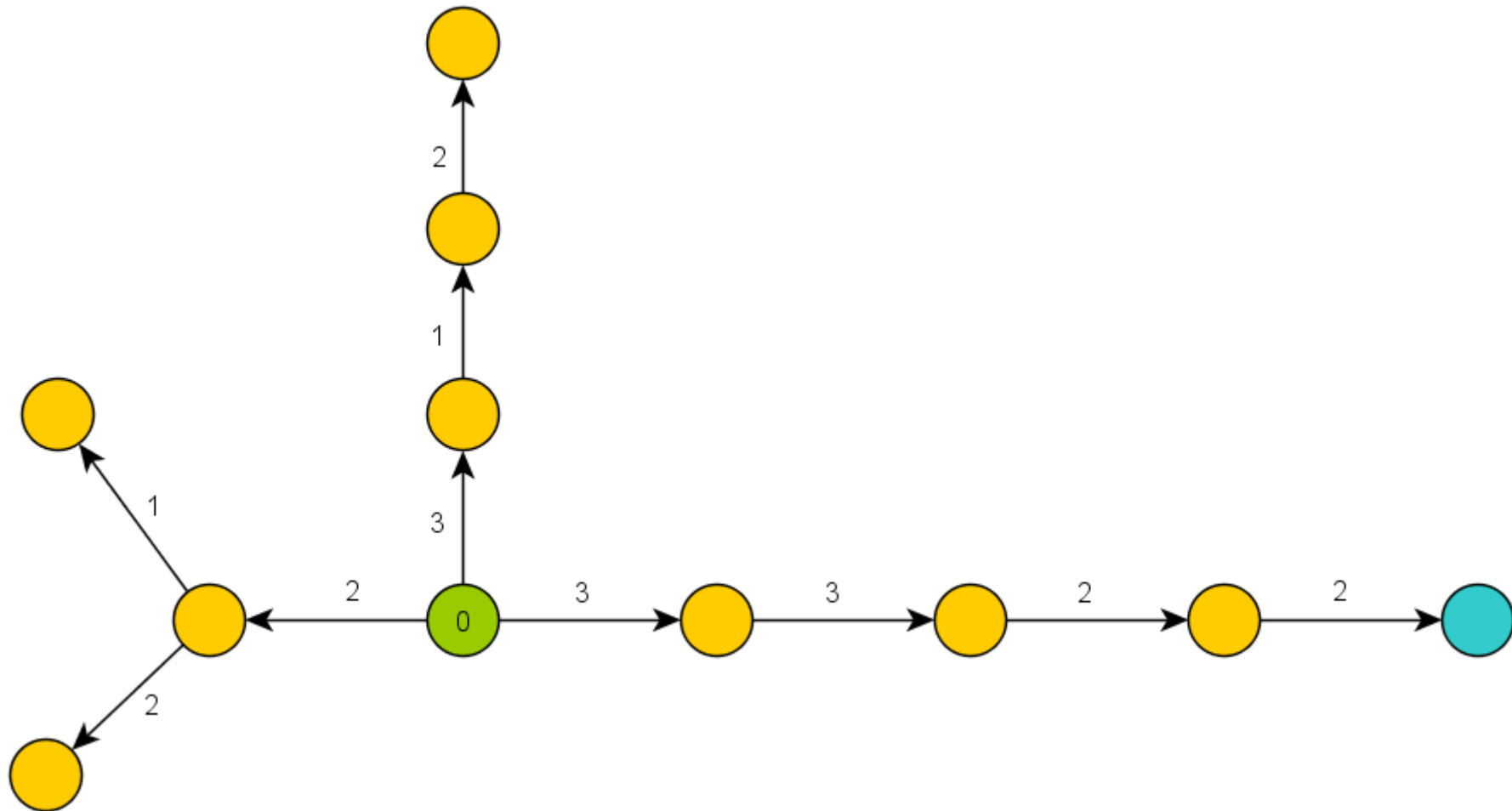
- Remembering Dijkstra's alg?
- Roughly speaking...

```
Nodes = {start}
while (!nodes.empty) {
    Node = pick_shortest_path(nodes)
    if (Node == Target) return
        reconstruct_path(Node)
    Nodes = Nodes \ Node
    Expand(Node, Nodes)
}
```



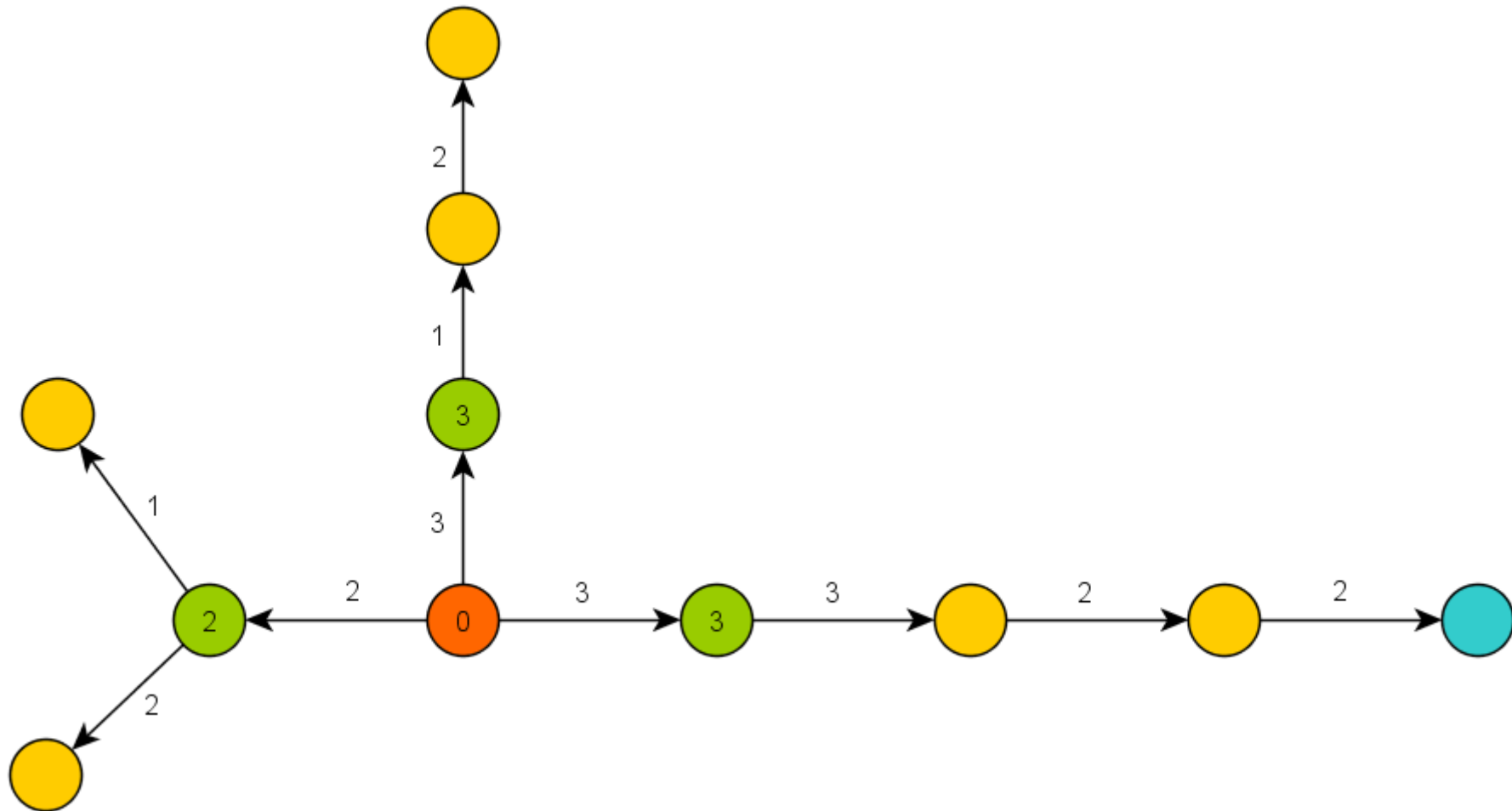
# A\* Algorithm

## Dijkstra Example I



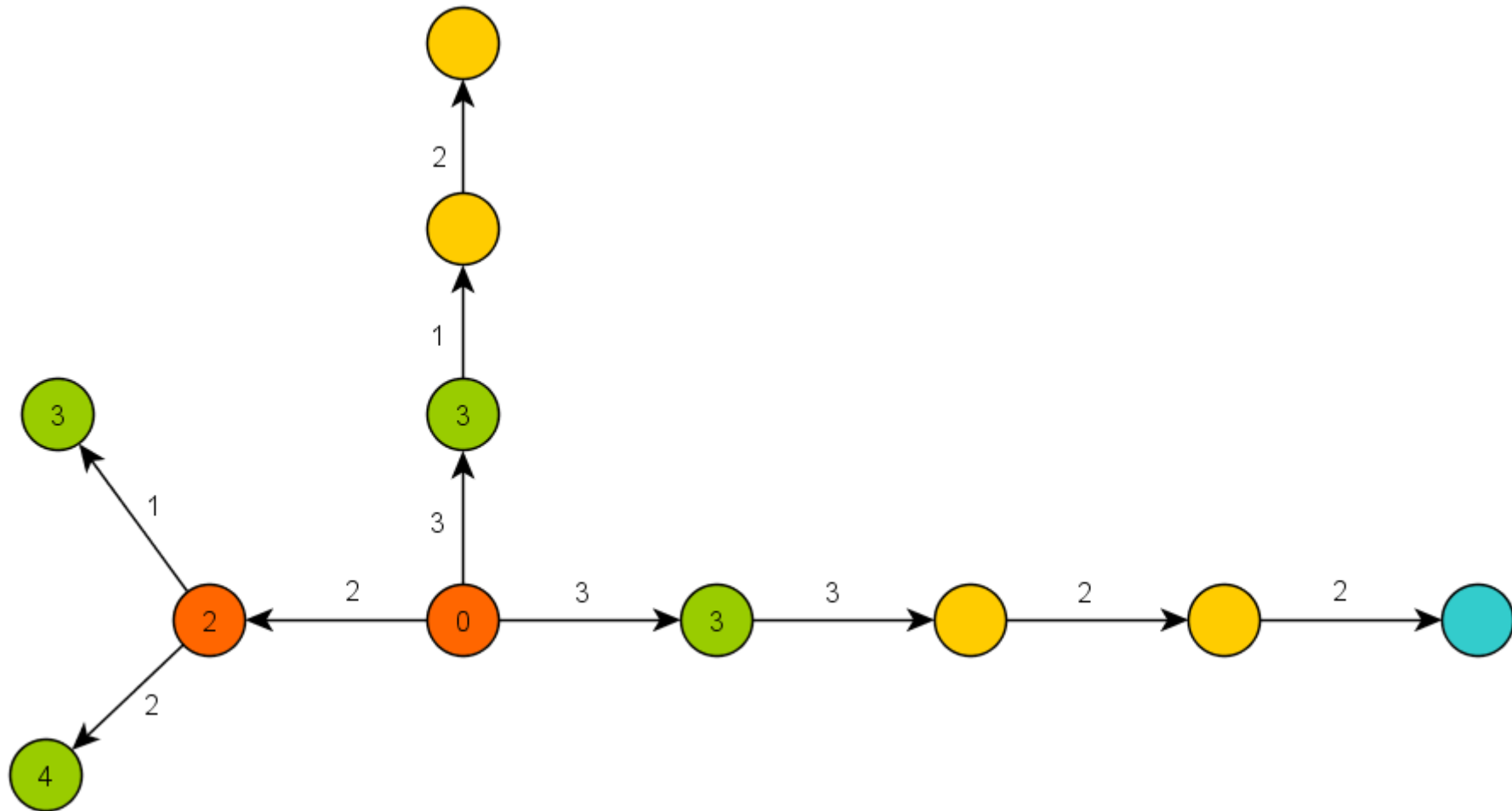
# A\* Algorithm

## Dijkstra Example II



# A\* Algorithm

## Dijkstra Example III



# A\* Algorithm

## Basics

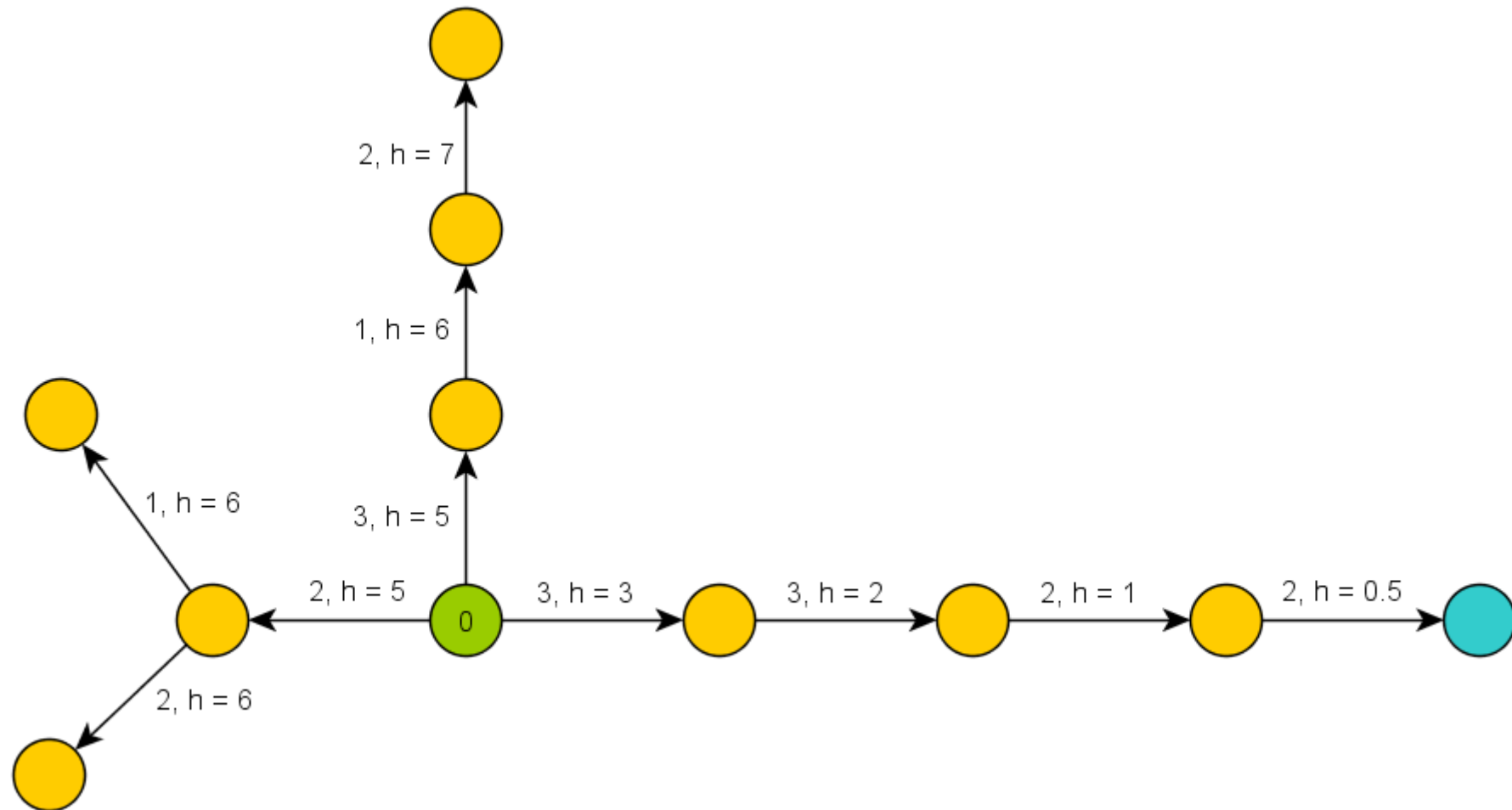


- A\* trick
- Roughly speaking...

```
Nodes = {start}
while (!nodes.empty) {
    Node = pick_the_most_promising(nodes)
    if (Node == Target) return
        reconstruct_path(Node)
    Nodes = Nodes \ Node
    Expand(Node, Nodes)
}
```

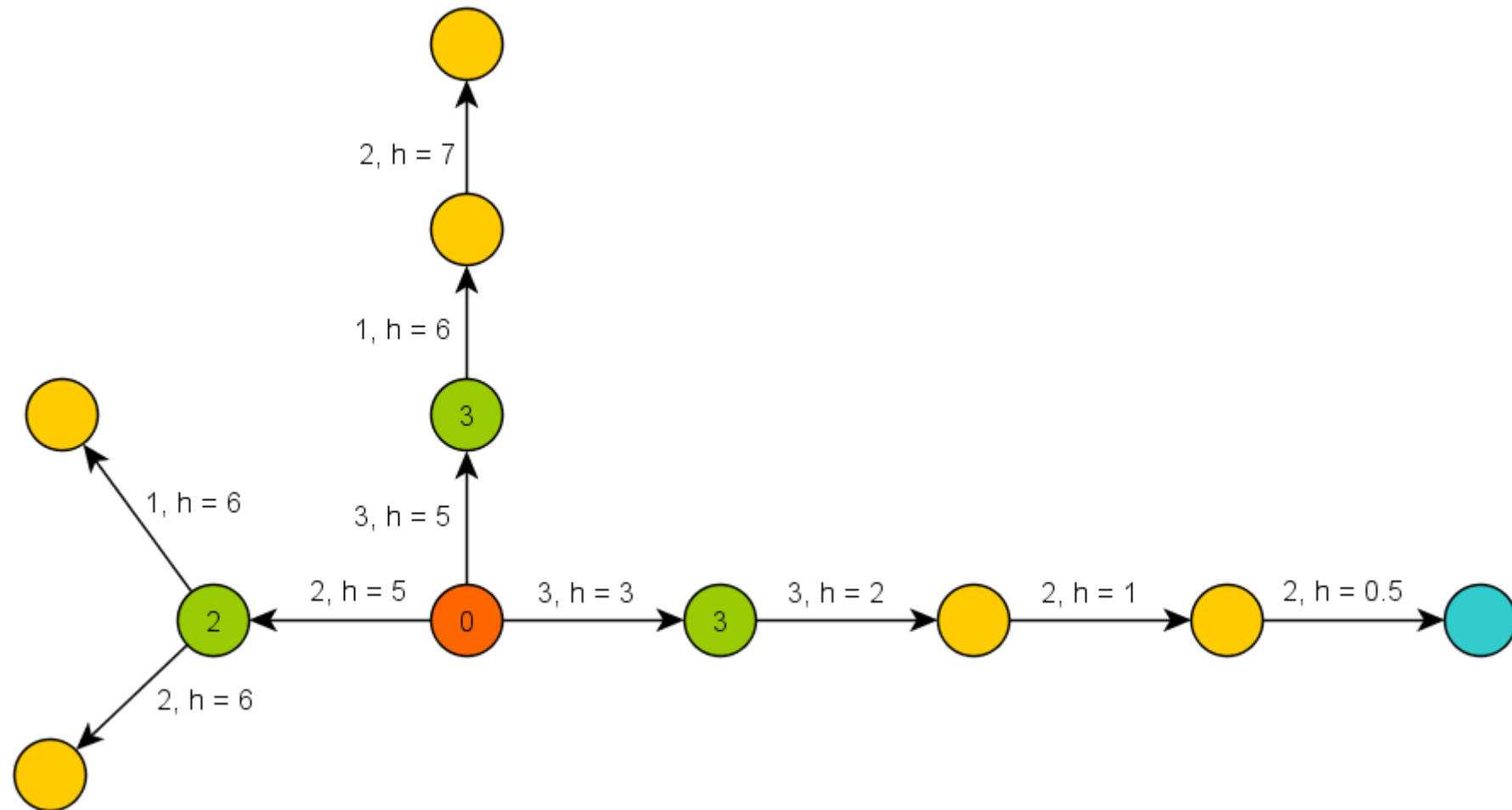
# A\* Algorithm

## A\* Example I



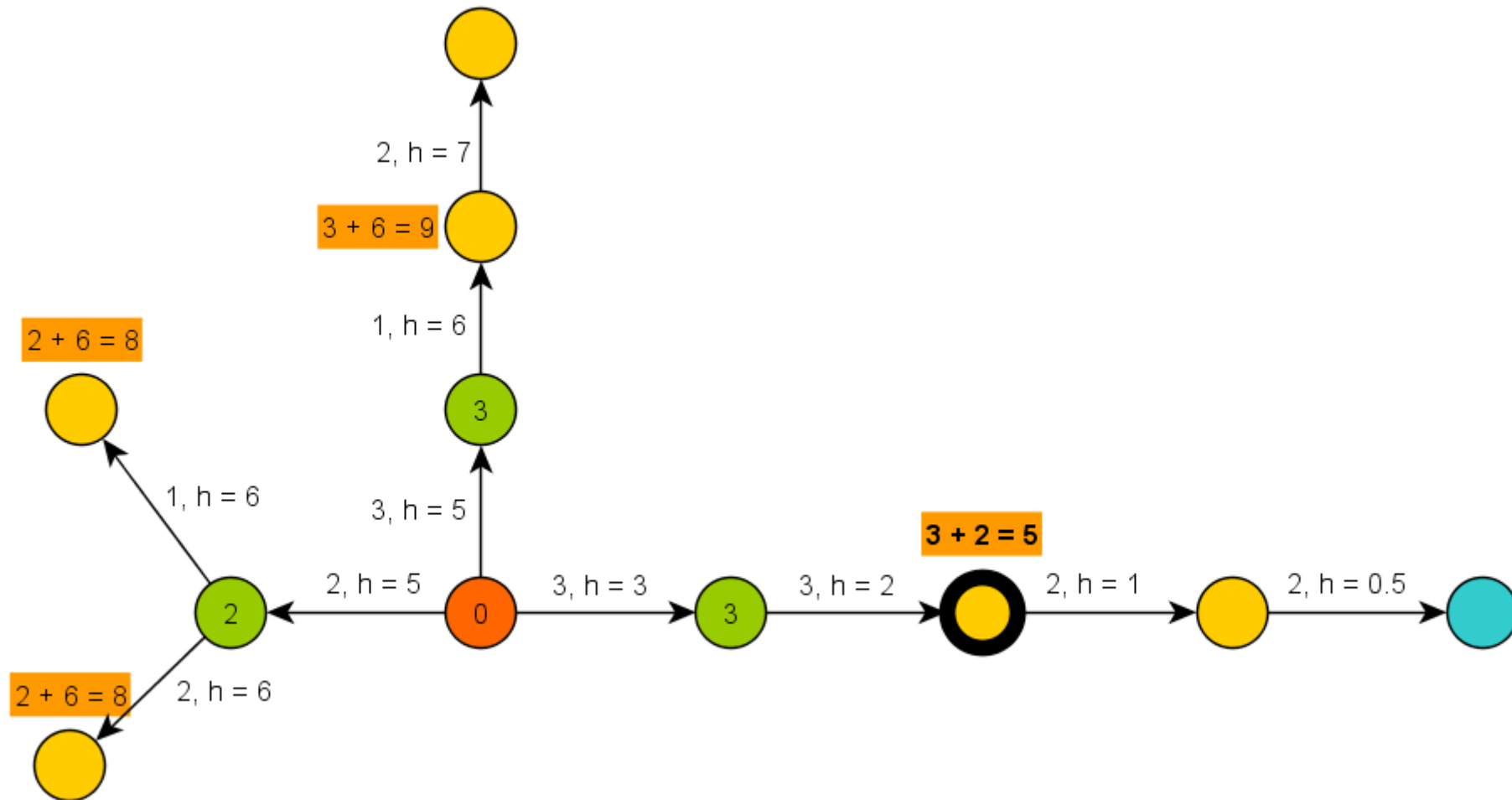
# A\* Algorithm

## A\* Example II



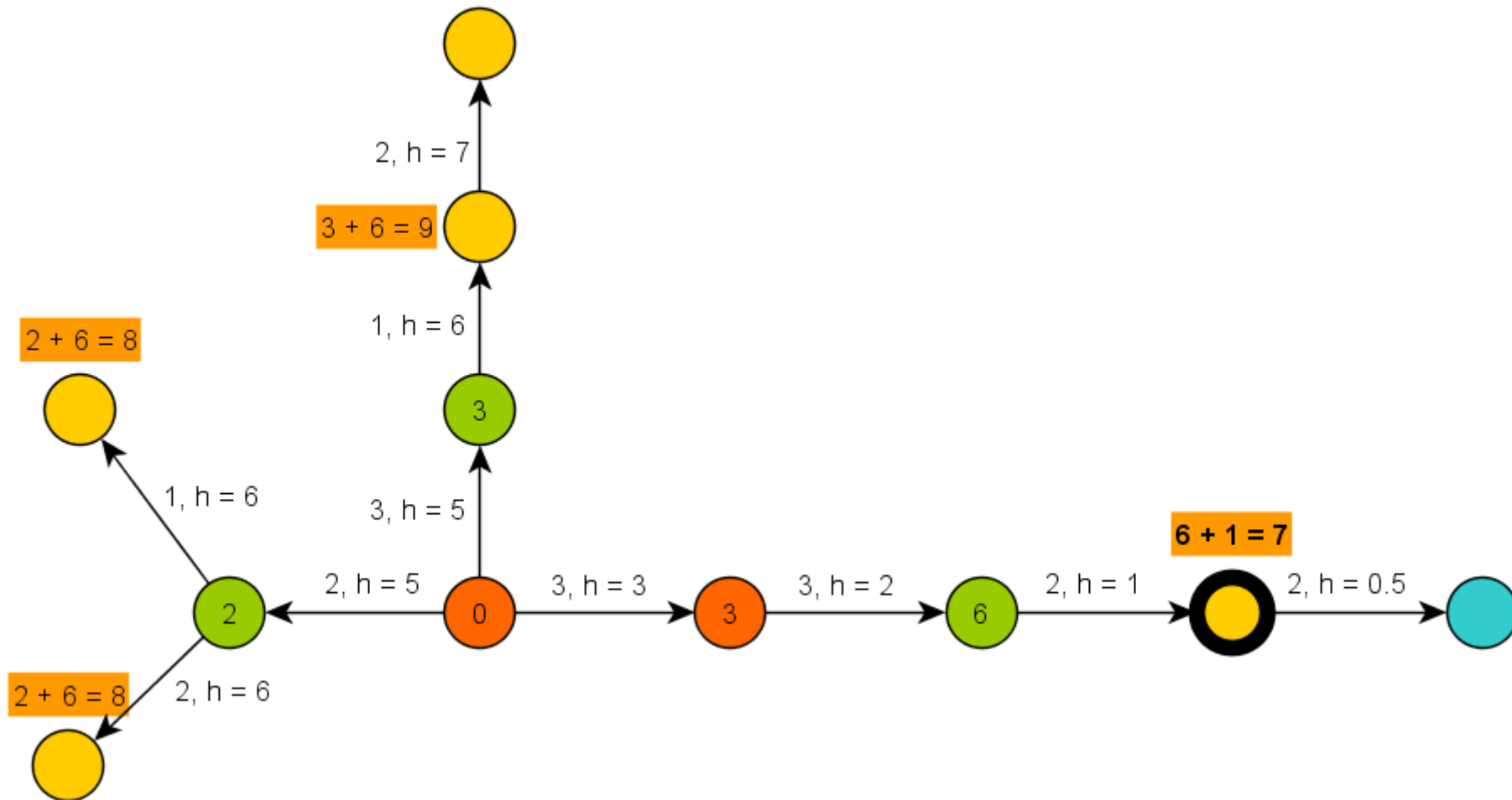
# A\* Algorithm

## A\* Example III



# A\* Algorithm

## A\* Example IV





# A\* Algorithm

## Basics



- A\* heuristic function must be... ?
  1. Admissible for correctness
    - Do not over-estimate the path-cost
  2. Consistent == Monotone (for efficiency)
    - “triangle inequation”
  
- Blah! Let’s hack it!
  - What if we impose additional COST to some nodes or links?

# A\* Algorithm

## Juggling with node/link costs



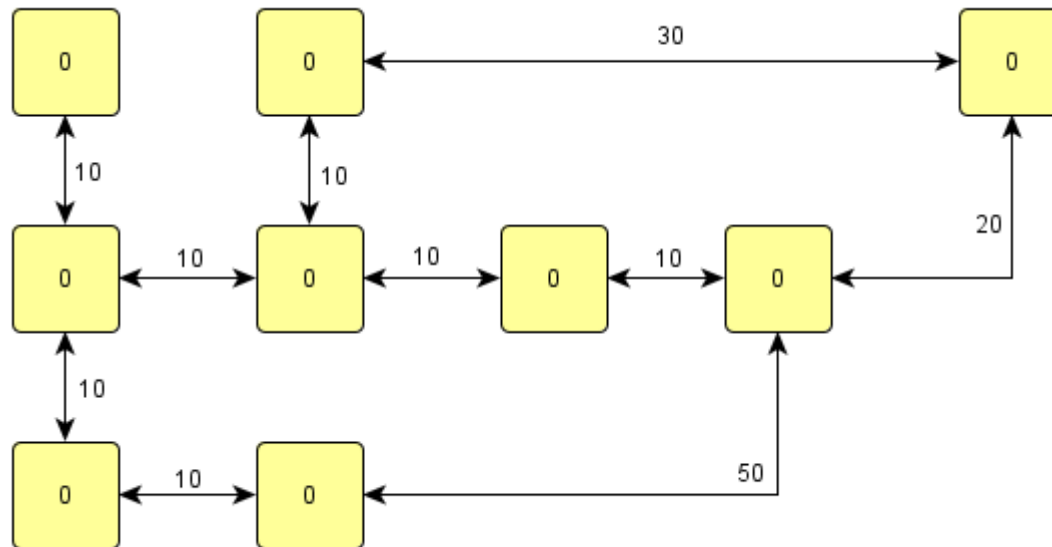
- $\text{Len}(\text{path})$  ... path length
  - $\text{min-Len-Path}(N, M)$  ... **shortest** path between N and M
  - B ... bad node/link B
  - EB ... extra cost visiting/traversing B
  - $\text{Cost}(\text{path})$  ... path cost (based on  $\text{Len}(\text{path})$ ) including EB
  - $\text{min-Cost-Path}(N, M)$  ... **the least cost** path between N and M
- 
- What  $\text{P-Len}(N, M)$  and  $\text{P-Cost}(N, M)$  look like?
    1.  **$\text{P-Len}(N, M) == \text{P-Cost}(N, M)$** 
      - There does not exist other path  $p(N, M)$  not-including B satisfying  $\text{Len}(p(N, M)) < \text{Len}(\text{P-Len}(N, M)) + \text{EB}$
    2.  **$\text{P-Len}(N, M) != \text{P-Cost}(N, M)$** 
      - We have found  $\text{Len}$ -longer path that does not traverse B satisfying  $\text{Len}(\text{P-Cost}(N, M)) < \text{Len}(\text{P-Len}(N, M)) + \text{EB}$

# A\* Algorithm

## Juggling with node/link costs



- Example map

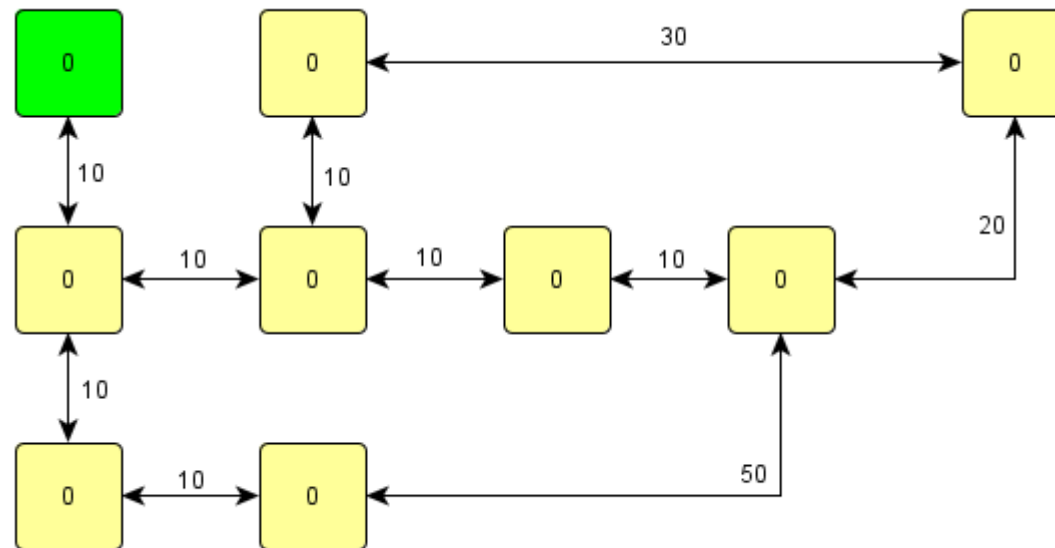


# A\* Algorithm

## Juggling with node/link costs



- Start-node

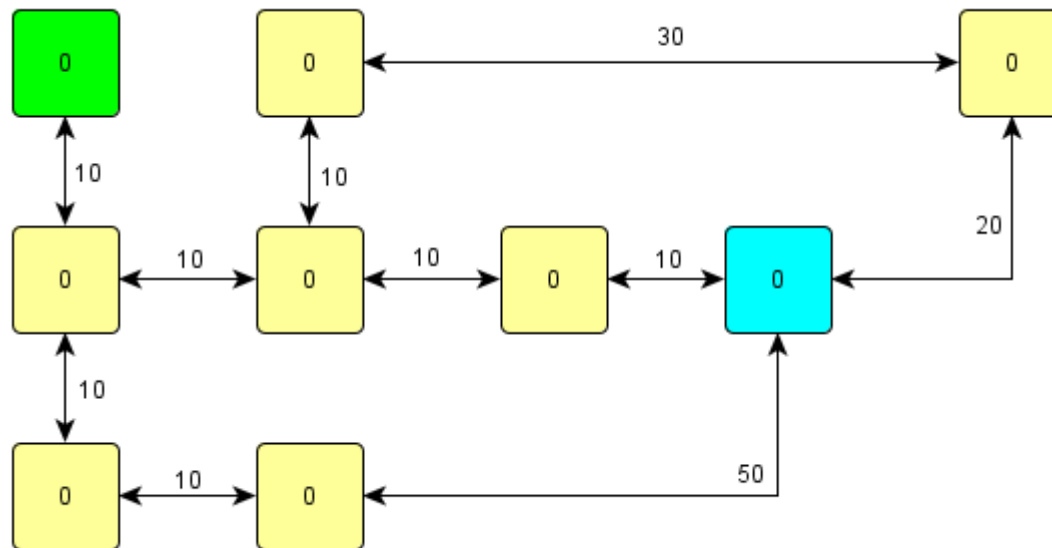


# A\* Algorithm

## Juggling with node/link costs



- Target-node

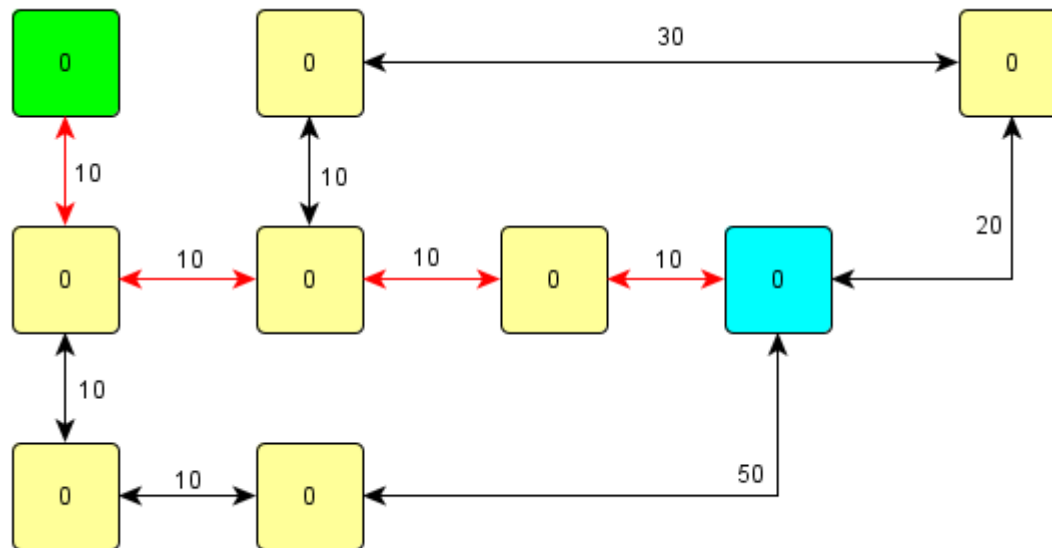


# A\* Algorithm

## Juggling with node/link costs



- Shortest path

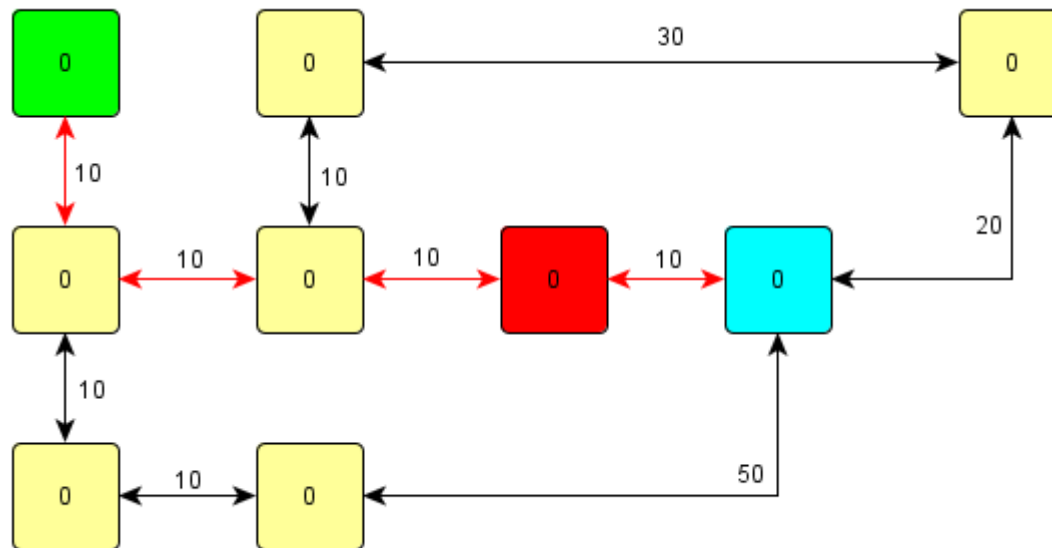


# A\* Algorithm

## Juggling with node/link costs



- Adversary we want to avoid

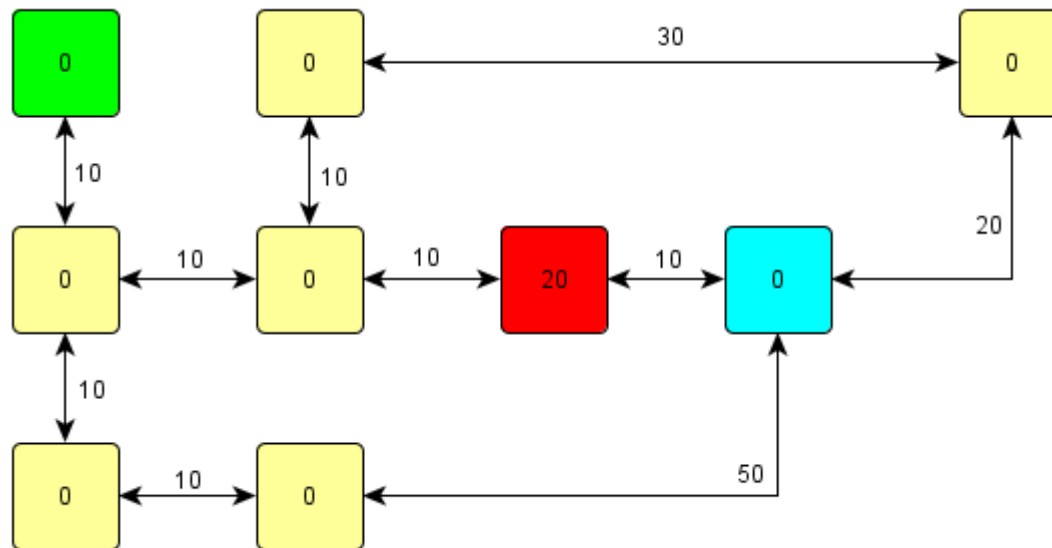


# A\* Algorithm

## Juggling with node/link costs



- Let's rise the NODE cost ... is it enough?



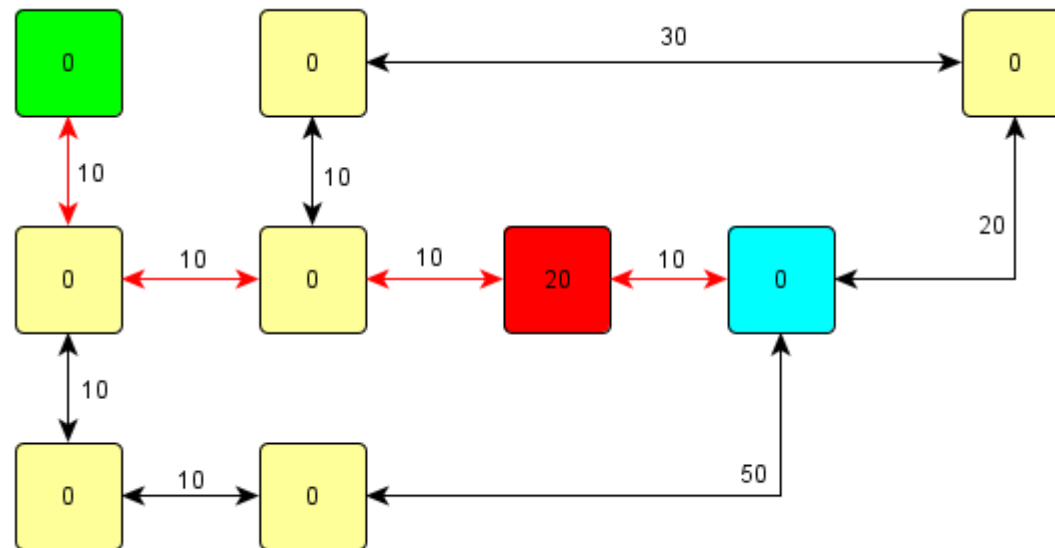


# A\* Algorithm

## Juggling with node/link costs



- No...

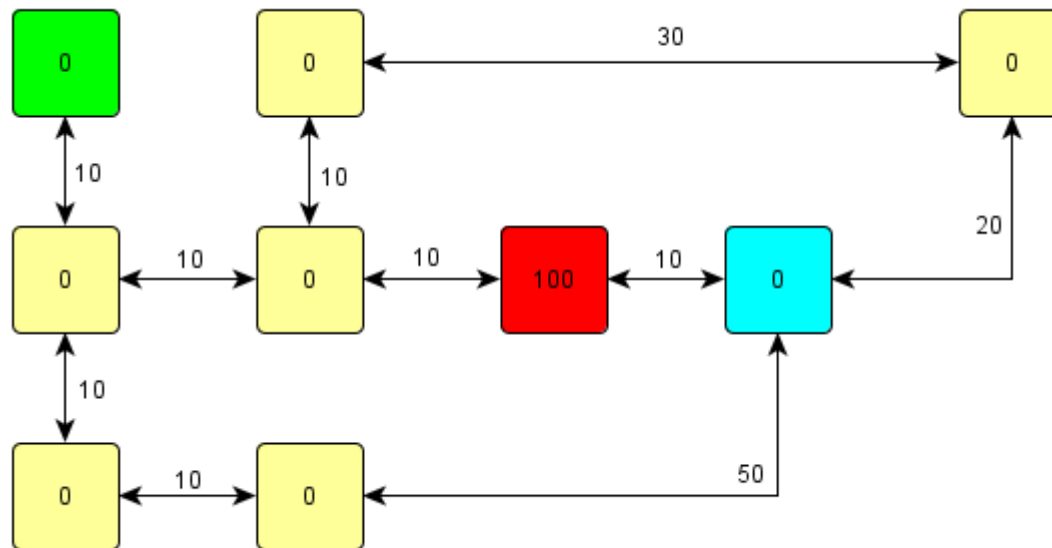


# A\* Algorithm

## Juggling with node/link costs



- Rise the NODE cost again... enough now?

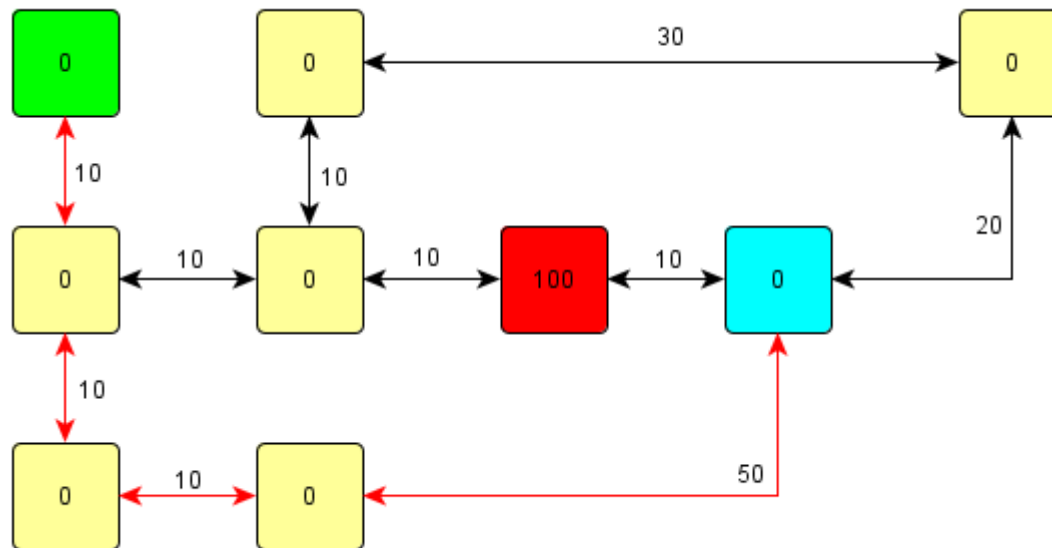


# A\* Algorithm

## Juggling with node/link costs



- Here you go!
  - Why was this path found?

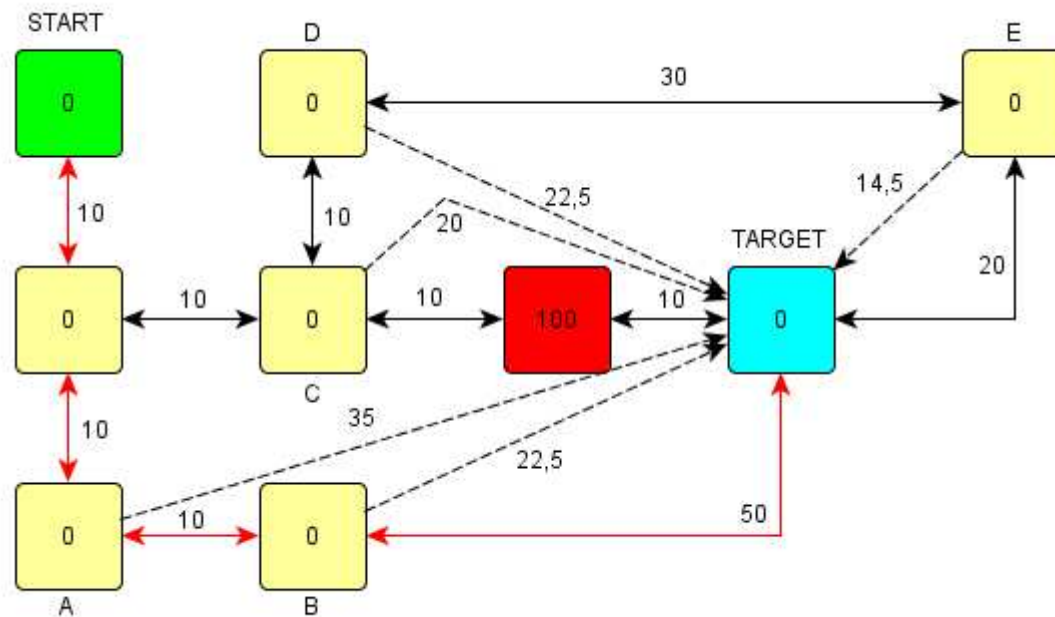


# A\* Algorithm

## Juggling with node/link costs



- Adding important heuristic costs
  - So, are we cheating or not?



# A\* Algorithm

## Map cost tricks



- Combine it with enemy position!
  - `extra cost = 500 / distance-to-enemy`
- Combine it with `Visibility` class!
  - `boolean Visibility.isVisible(ILocated, ILocated)`
- Combine both enemy position and the visibility!
- Combine with already-found path + `fwMap` and find different paths!
- Play with the cost iteratively
  - Different path not found? Ok, just rise the cost...
  - Does different path even exist?
    - ⇒ Try to “forbid” some node/link completely

# A\* Algorithm

## Pogamut 3 Classes



- **UT2004AStar**

```
this.aStar.findPath( from, to, IPFMapView );
```

- **Implement your own custom IPFMapView:**

```
new IPFMapView<NavPoint>() {  
  
    public int getNodeExtraCost(NavPoint node, int mapCost) {}  
  
    public int getArcExtraCost(NavPoint nodeFrom, NavPoint nodeTo, int mapCost) {}  
  
    public Collection<NavPoint> getExtraNeighbors(NavPoint node,  
                                                  Collection<NavPoint> mapNeighbors) {}  
  
    public boolean isNodeOpened(NavPoint node) {}  
  
    public boolean isArcOpened(NavPoint nodeFrom, NavPoint nodeTo) {}  
  
}
```

# Today's menu



1. Big Picture
2. Visibility abstraction
  - Visibility matrix
  - Visibility
  - `this.visibility`
3. How to reason about path
  - A\* and custom map view
  - `UT2004AStar, IPFMapView<NavPoint>`
  - `this.aStar`
4. **Hide&Seek Game**
  - **Rules, Map**
  - `HideAndSeekMap`
5. Hide&Seek Tournament Announcement

# Hide&Seek Game

## Children play



- Custom “game-mode” for UT2004
- Two roles:
  1. Seeker (having “it”)
  2. Runner
- Seeker has to find runners and then get home (safe point) first to “capture them”
- Runners have to make it home (to safe point) before Seeker
- `this.hide` agent module
- Custom map: DM-HideAndSeekMap



# Hide&Seek Game

## Rules specifics



- One match = 3 games of 10 rounds each of hide and seek with fixed seeker for each game
  - 1 round = 60 seconds (first 8 seconds hide time, next 5 seconds restricted safe area time)
- Spotting
  - Seeker “spots” runner when he sees him for at least 600 ms (cca “two logic() ticks”)
  - Seeker is spawned into the map after first 8 seconds
- Safe area
  - Runners are not allowed to dwell around safe point for certain amount of time at the beginning of the game (5 seconds)

# Hide&Seek Game

## Task point rewards



### ■ Scoring RUNNER

- Runner captured by seeker **-10**
- Runner fouled (went into safe area before timeout) **-1000**
- Runner made it to safe area before seeker **150**
- Runner survived round (haven't been captured by seeker) **50**

### ■ Scoring SEEKER

- Seeker captured runner (spotted + made it to s. a. first) **250**
- Runner spotted **50**
- Runner escaped (made it to safe area before seeker) **-20**
- Runner survived (neither of them made it to safe area) **-10**

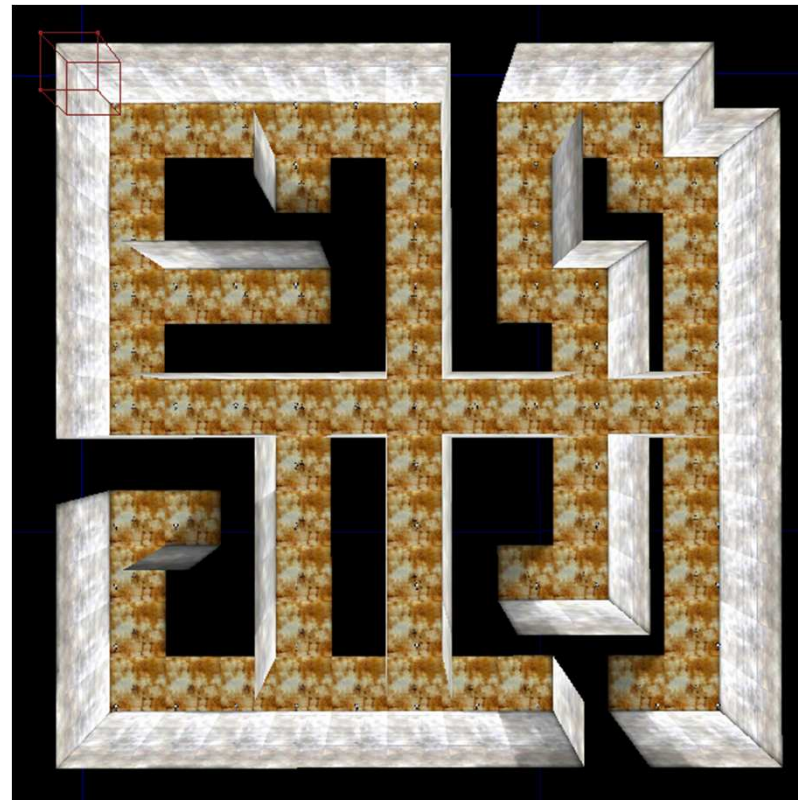
# Hide&Seek Game

## Custom map



- DM-HideAndSeekMap

```
#####  
#.....#...##  
#...#.#.#.#.#  
#...###.#...#  
#.....#.#.#.#  
#...###.###.#.#  
#.....#  
#####.#...#.#.#  
#..#.#.#.#.#.#  
#...#.#.#.#.#  
#...#.#.#####.#  
#.....#...#  
#####
```



# Today's menu



1. Big Picture
2. Visibility abstraction
  - Visibility matrix
  - Visibility
  - `this.visibility`
3. How to reason about path
  - A\* and custom map view
  - `UT2004AStar, IPFMapView<NavPoint>`
  - `this.aStar`
4. Hide&Seek Game
  - Rules, Map
  - `HideAndSeekMap`
5. **Hide&Seek Tournament Announcement**

# Hide&Seek Tournament

Chance to score extra points!



- 3 bots
  - 1 Seeker, 2 Runners
- Random groups, Fixed map
- Fixed Seeker - 3 matches per group
- Only bots submitted until Friday 19.4.2013, 23:59 will participate
- No shooting allowed, no bot speed reconfigurations allowed, no manual respawns allowed

# Assignment 6

## Hide&Seek Bot



- Create **Hide&Seek Bot**
  - Implement both Seeker and Runner
  - Tournament will be played on a different map, so we do not recommend using “static” information e.g. run to [1000,200,100] 😊
  - To run the hide and seek match launch **HideAndSeekGame** class!
  - For the tournament name the bot with your name in **getInitializeCommand()** method

# Send us finished assignment

Via e-mail:

- *Subject*
  - "Pogamut homework 2013 – Assignment X"
    - Replace 'x' with the assignment number and the subject has to be without quotes of course
    - ...or face **-2 score penalization**
- *To*
  - [jakub.gemrot@gmail.com](mailto:jakub.gemrot@gmail.com)
    - Jakub Gemrot (Monday practice lessons)
  - [michal.bida@gmail.com](mailto:michal.bida@gmail.com)
    - Michal Bida (Thursday practice lessons)
- *Attachment*
  - Completely zip-up your project(s) folder except 'target' directory and IDE specific files (or face **-2 score penalization**)
- *Body*
  - **Please send us information about how much time it took you to finish the assignment + any comments regarding your implementation struggle**
    - *Information won't be abused/made public*
    - *In fact it helps to make the practice lessons better*
  - Don't forget to mention your full name!

# Questions?

I sense a soul in search of answers...

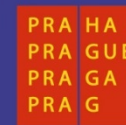


- We do not own the patent of perfection (yet...)
- In case of doubts about the assignment, tournament or hard problems, bugs don't hesitate to contact us!
  - Jakub Gemrot (Monday practice lessons)
    - [jakub.gemrot@gmail.com](mailto:jakub.gemrot@gmail.com)
  - Michal Bída (Thursday practice lessons)
    - [michal.bida@gmail.com](mailto:michal.bida@gmail.com)





OPERAČNÍ PROGRAM PRAHA  
ADAPTABILITA



EVROPSKÁ UNIE

# DĚKUJI ZA POZORNOST



Evropský sociální fond  
Praha & EU: Investujeme do vaší budoucnosti