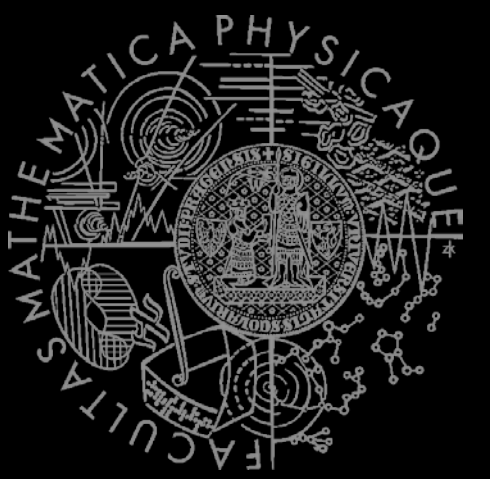
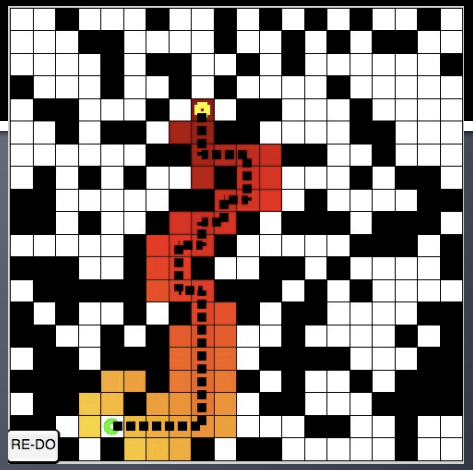
Faculty of mathematics and physics
Charles University in Prague
5th April 2016



UT2004 bots made easy!

Pogamut 3

Lecture 6 – A* + Visibility



Bussiness as usual

- Copy UT2004 into D:\
- Start downloading the bot:
  - **http://alturl.com/45x9m**
  - http://diana.ms.mff.cuni.cz/pogamut_files/lectures/2014-2015/L6-HideAndSeekBot.zip

Warm Up!

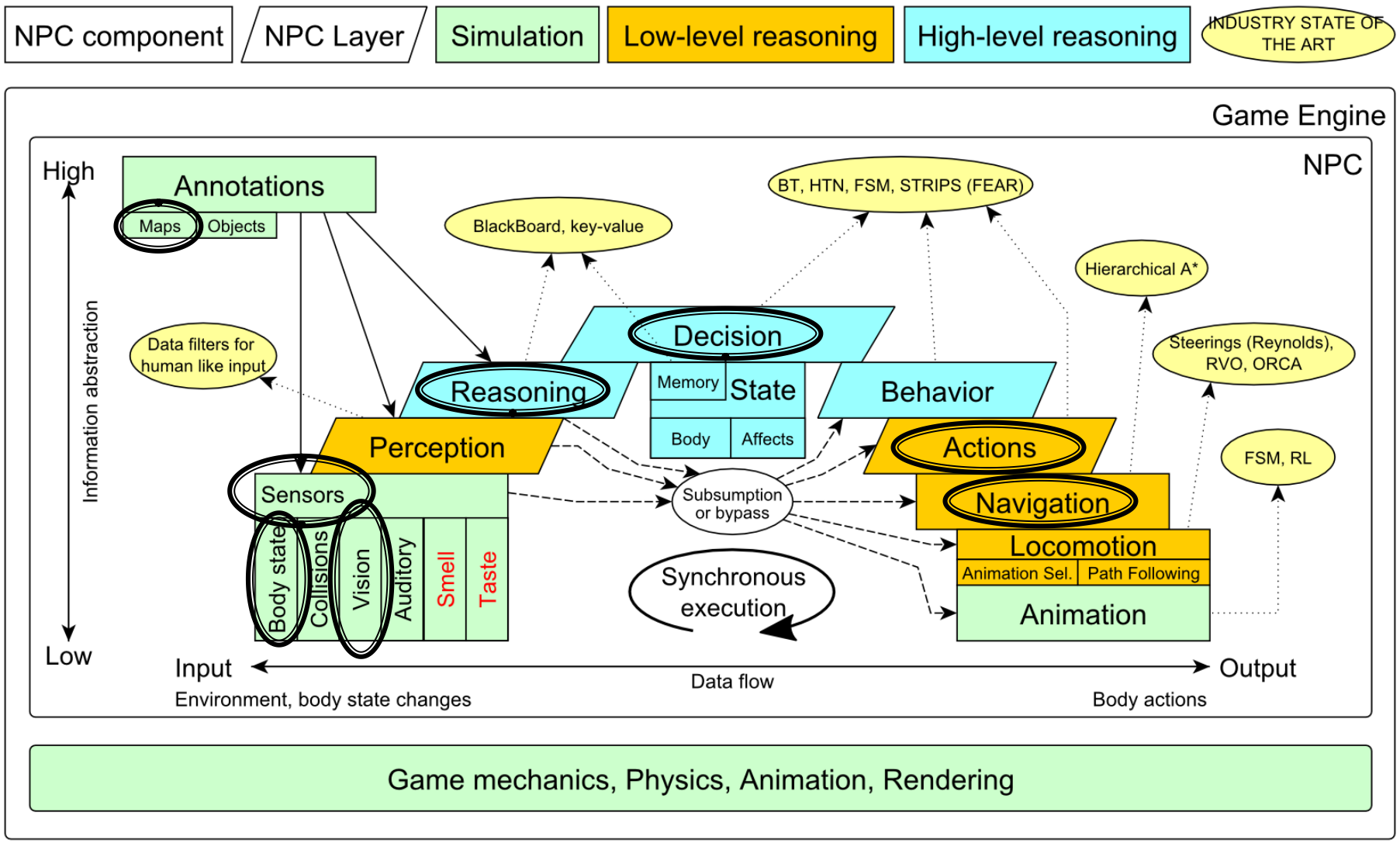
- Fill the short test for this lessons
  - 7 minutes
  - **https://goo.gl/kgs0ly**
  - `O vs 0, i vs. l vs. 1`

  - https://docs.google.com/forms/d/1djQW1MCz2dCX3rMzl6OvsBnz-Kq1UAyv_mpiwqG7dbA/viewform

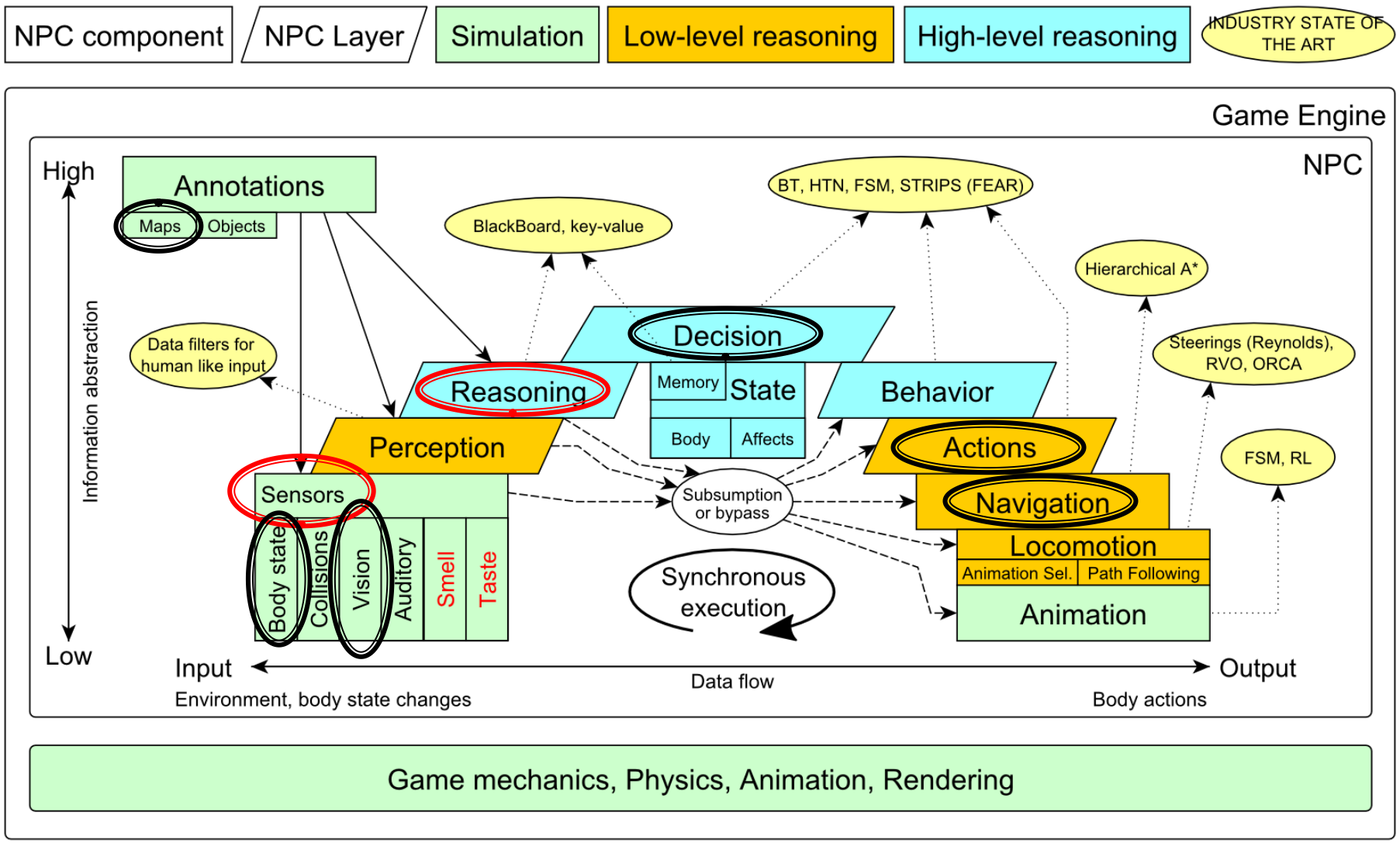
Today's menu

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



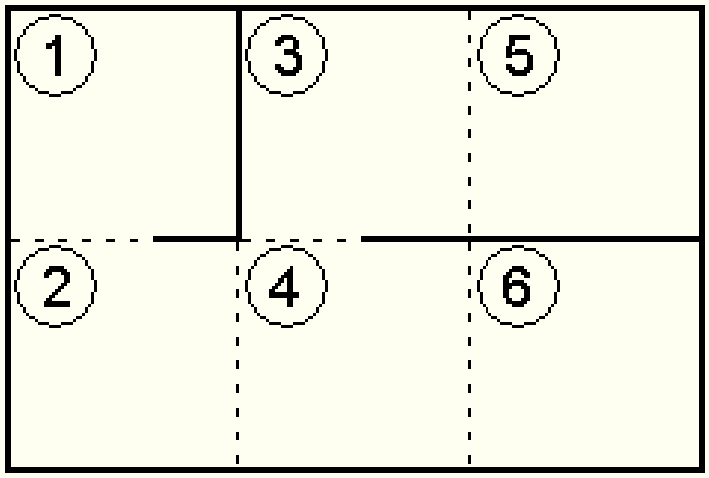
Today's menu

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

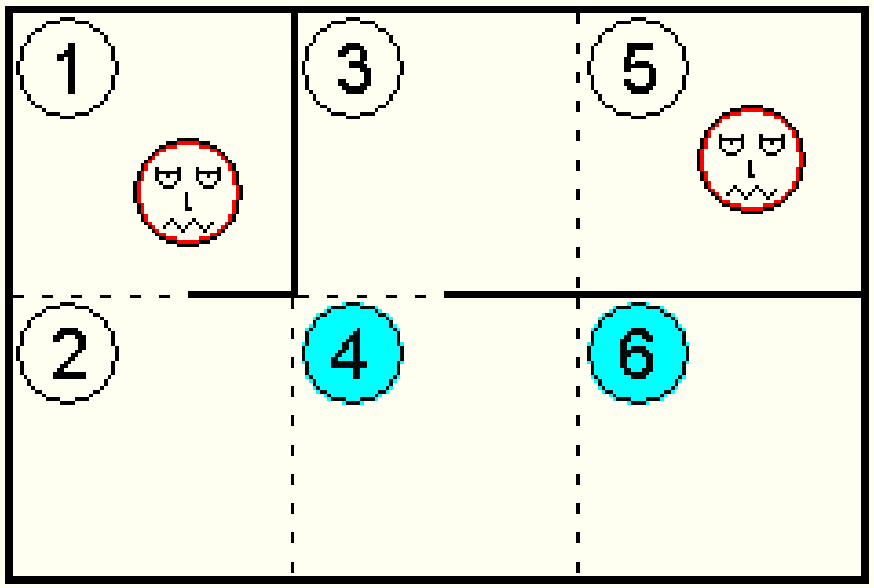


	1	2	3	4	5	6
V_1	1	1	0	0	0	0
V_2	1	1	0	1	0	1
V_3	0	0	1	0	1	0
V_4	0	1	0	1	0	1
V_5	0	0	1	0	1	0
V_6	0	1	0	1	0	1

Visibility Matrix

How to get to cover?

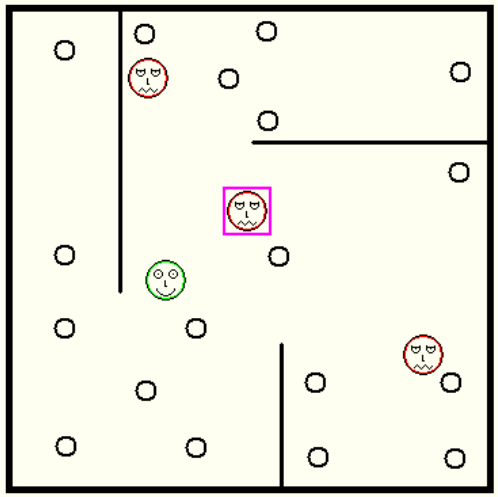
- How to find the cover?
  - Enemies … $E_1..E_k$
  - Safe waypoints … $S = \neg \bigvee_{i=1}^{k} V_{E_i}$



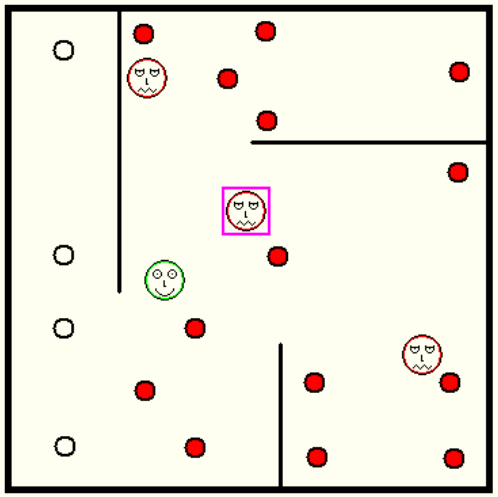
	1	2	3	4	5	6
V_1	1	1	0	0	0	0
V_2	1	1	0	1	0	1
V_3	0	0	1	0	1	0
V_4	0	1	0	1	0	1
V_5	0	0	1	0	1	0
V_6	0	1	0	1	0	1

Visibility Matrix

Smart attack

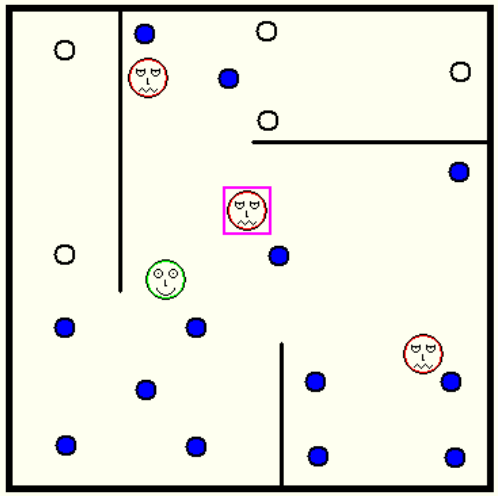


1. Choose target T
2. Others are enemies E_i



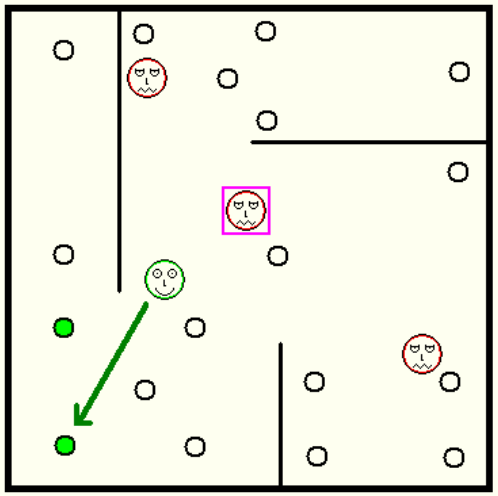
3. Navpoints other enemies E_i can see

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



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 (this.visibility)

  `getNearestVisibilityLocationTo``(ILocated)`

  `getCoverPointsFrom``(ILocated)`

  `getCoverPointsFromN``(ILocated…)`

  `getMatrix``()`

  `isVisible(ILocated, ILocated)`

- `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

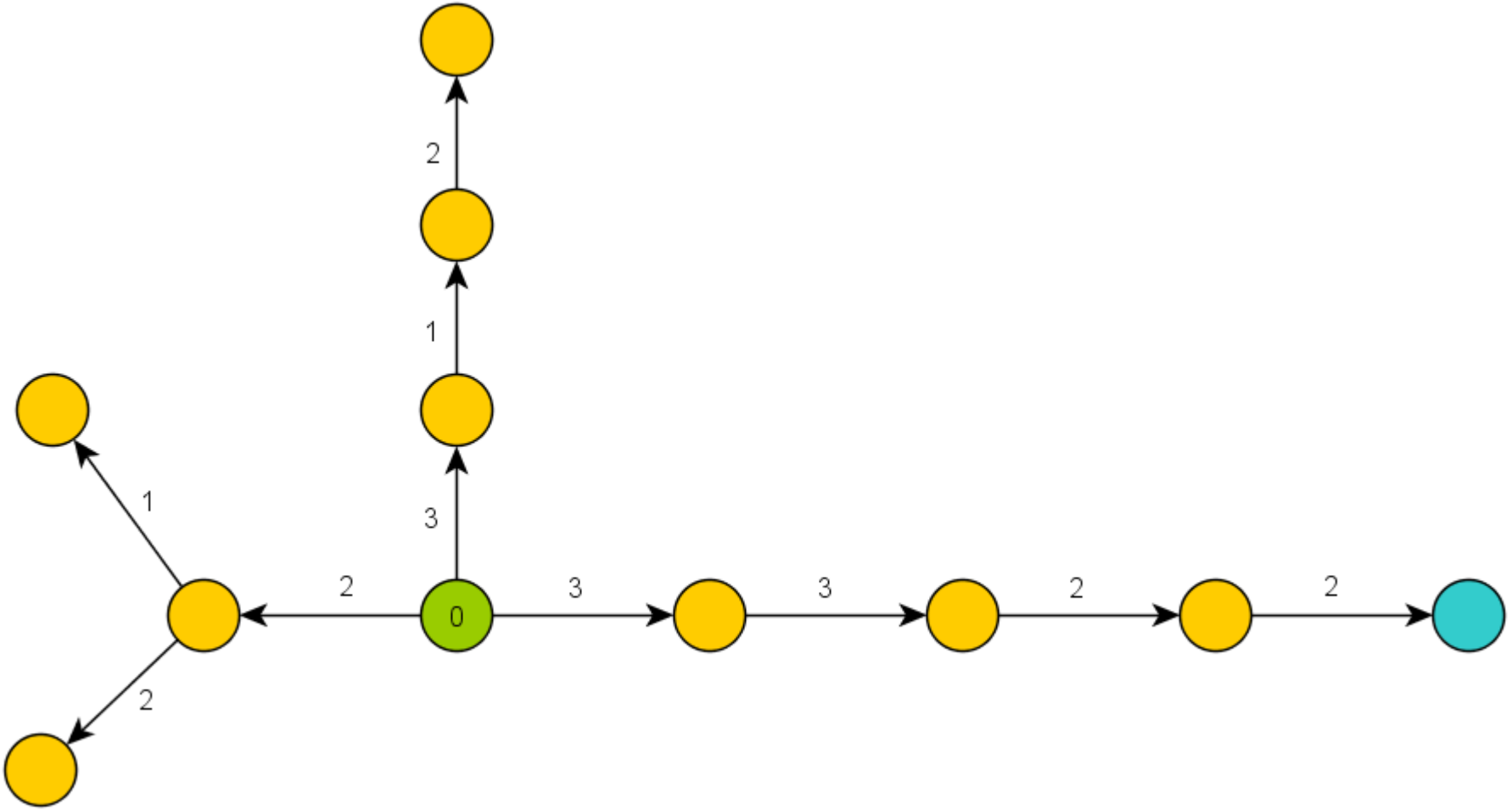
Dijkstra

- 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)
}
```

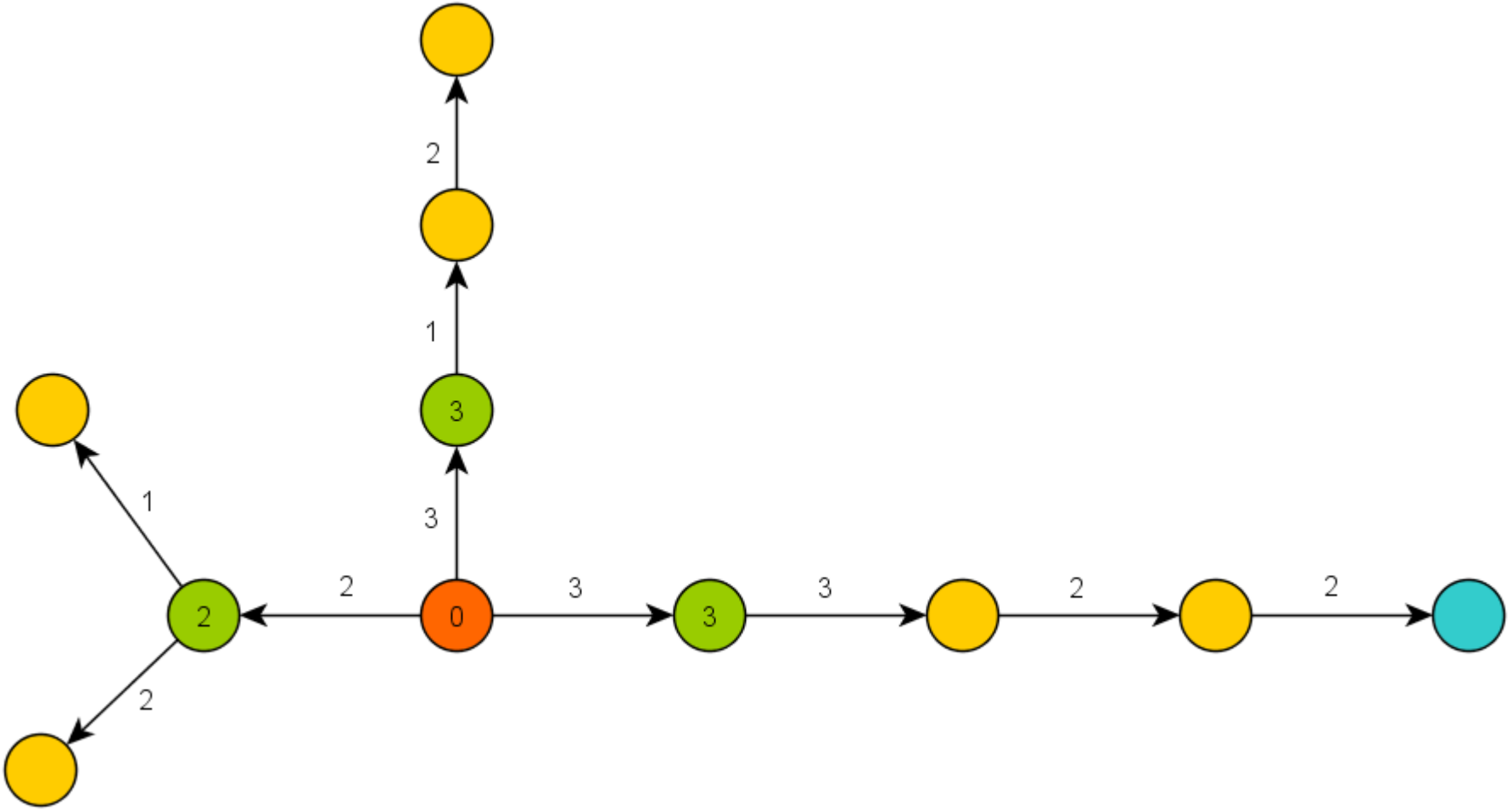
A* Algorithm

Dijkstra Example I



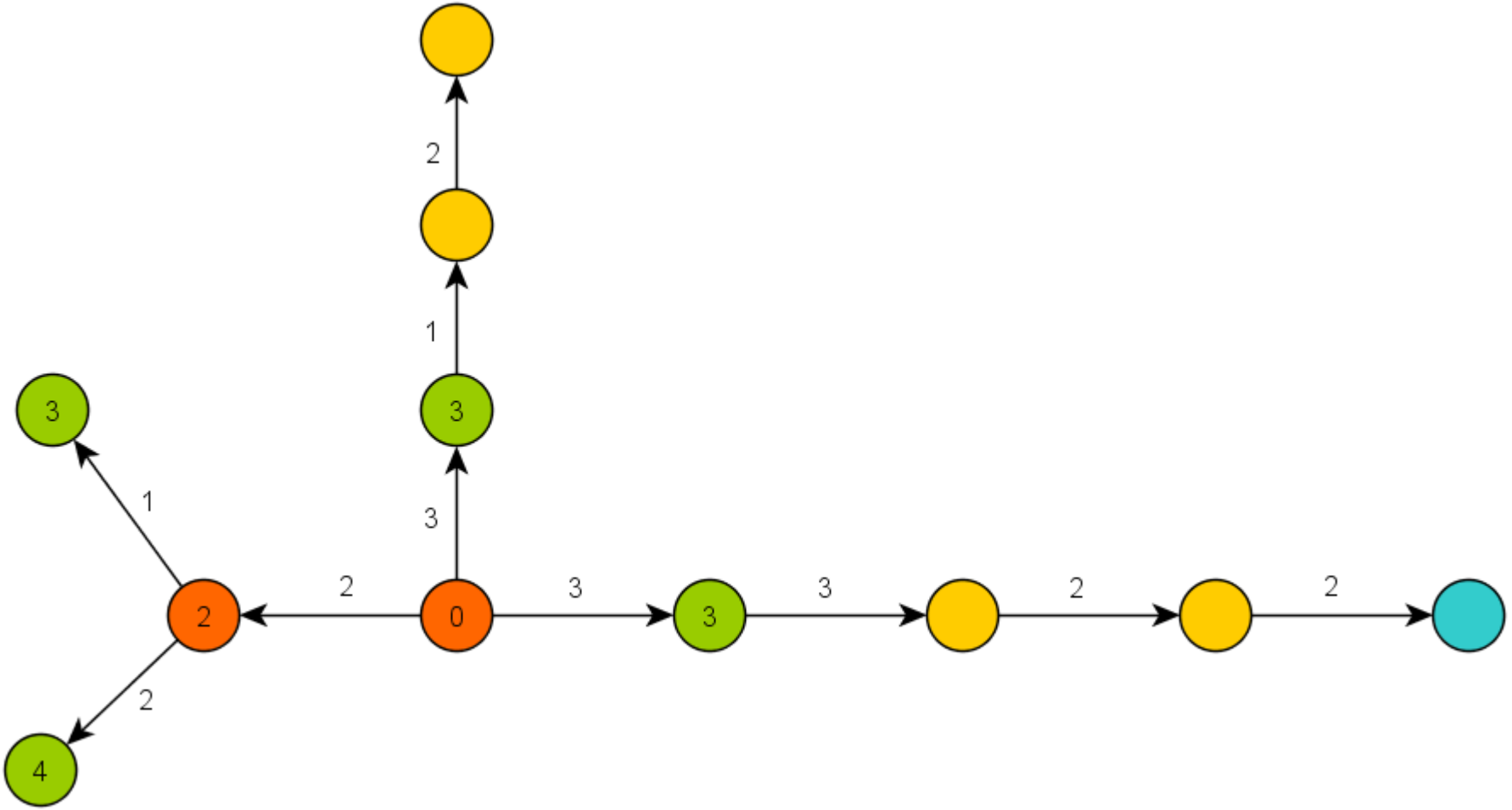
A* Algorithm

Dijkstra Example II



A* Algorithm

Dijkstra Example III



A* Algorithm
Basics

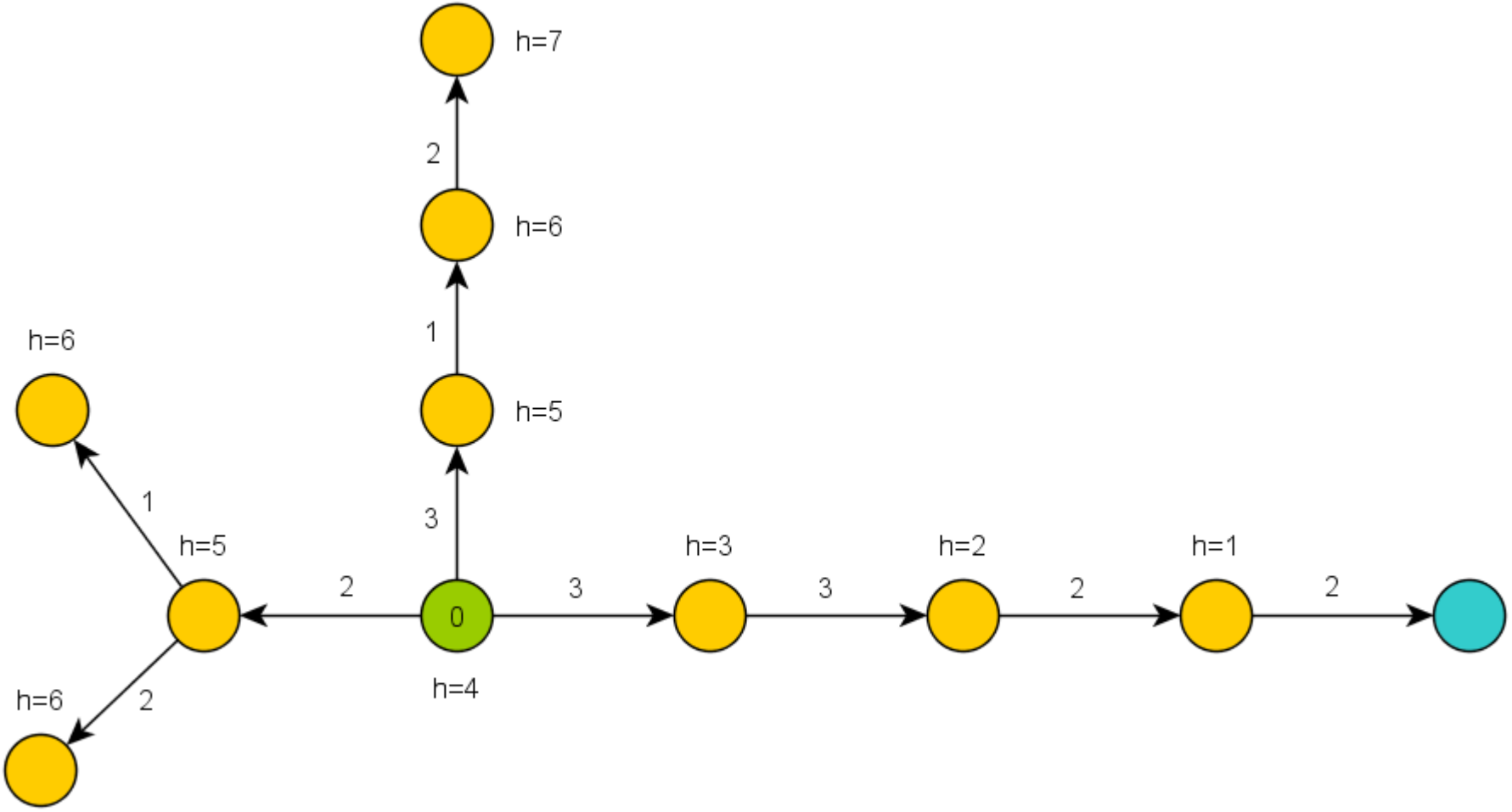
- A* trick

- 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)
}
```

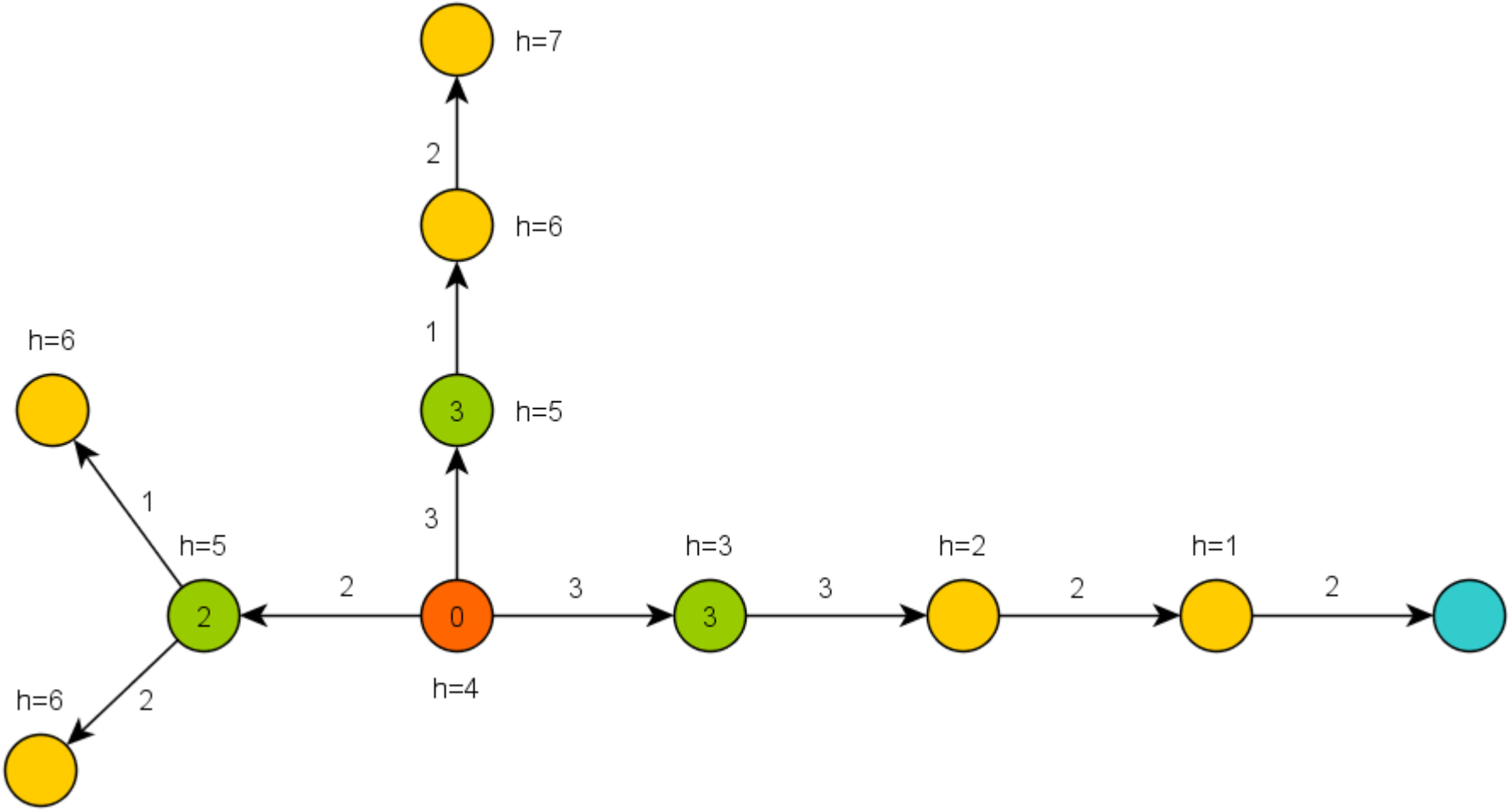
A* Algorithm

A* Example I



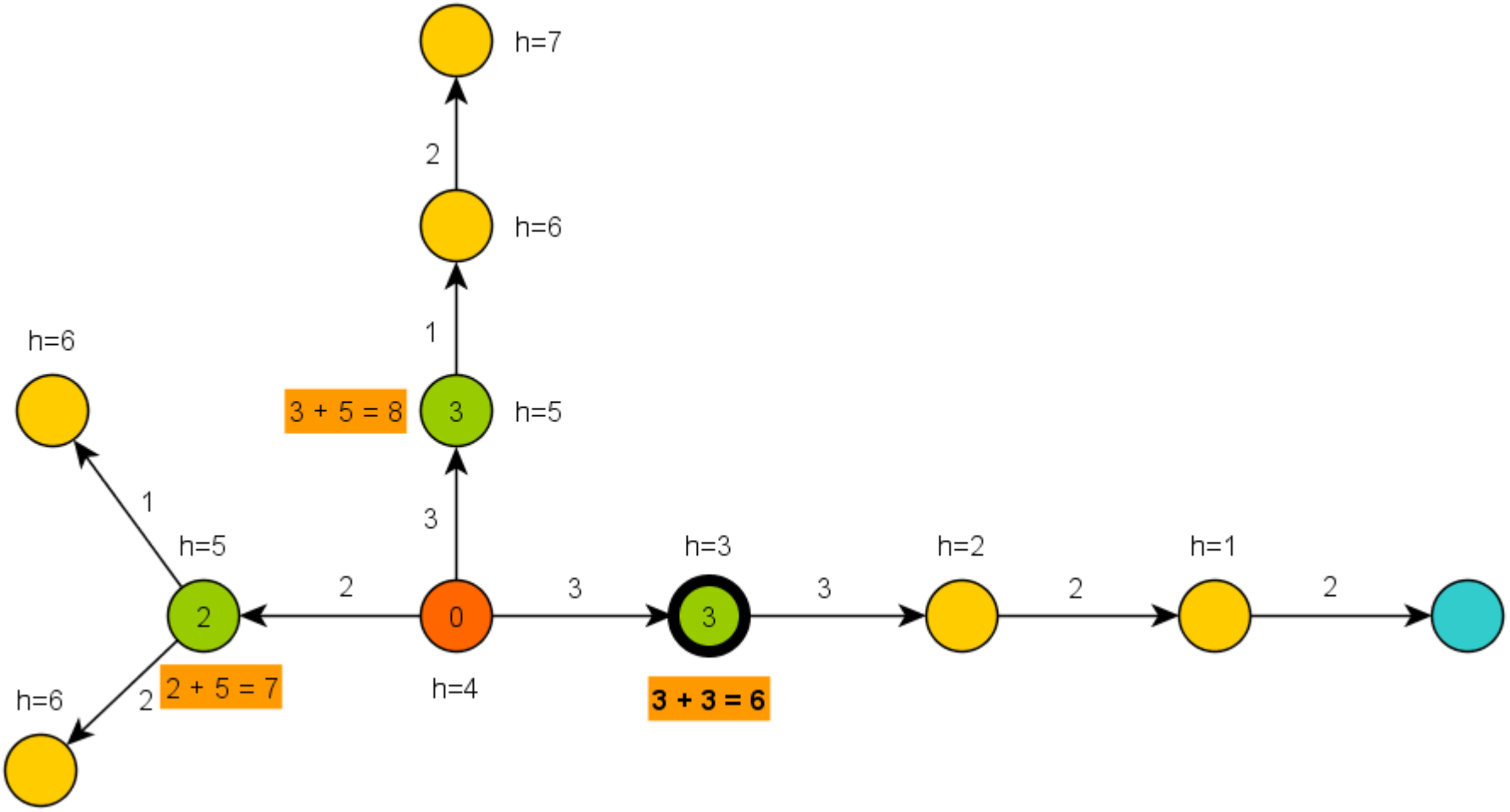
A* Algorithm

A* Example II



A* Algorithm

A* Example III



A* Algorithm
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

- Let's choose some "nodes" or "links" that we want to avoid
  - `B` … BADDIES … nodes or links with extra cost
  - `EC(B)` … EXTRA COST … sum of extra cost over the B set
- We then have two types of metrics for the path
  - `Len(p)` … PATH LENGTH … real environment path length
  - `Cost(p)` … PATH COST … `Len(p) + EC(p)`
- Thus we can run A* using those two metrics
  - `A*-Len(N,M)` … outputs the shortest path between nodes N and M
  - `A*-Cost(N,M)` … outputs the least costly path between nodes N and M
- What do `A*-Len(N,M)` and `A*-Cost(N,M)` look like?

A* Algorithm

Juggling with node/link costs

- Let's choose some "nodes" or "links" that we want to avoid
  - `B` … BADDIES … nodes or links with extra cost
  - `EC(B)` … EXTRA COST … sum of extra cost over the B set
- We then have two types of metrics for the path
  - `Len(p)` … PATH LENGTH … real environment path length
  - `Cost(p)` … PATH COST … `Len(p) + EC(p)`
- Thus we can run A* using those two metrics
  - `A*-Len(N,M)` … outputs the shortest path between nodes N and M
  - `A*-Cost(N,M)` … outputs the least costly path between nodes N and M
- What do `A*-Len(N,M)` and `A*-Cost(N,M)` look like?

1. `A*-Len(N,M) != A*-Cost(N,M)`
   - `A*-Len(N,M)` path contains some `B'` that are not on the path of `A*-Cost(N,M)`

⇒ We have found a detour that is shorter than `EC(B')`!

- `Cost(A*-Cost(N,M)) < Len(A*-Len(N,M)) + EC(A*-Len(N,M))`

A* Algorithm

Juggling with node/link costs

- Let's choose some "nodes" or "links" that we want to avoid
  - `B` … BADDIES … nodes or links with extra cost
  - `EC(B)` … EXTRA COST … sum of extra cost over the B set
- We then have two types of metrics for the path
  - `Len(p)` … PATH LENGTH … real environment path length
  - `Cost(p)` … PATH COST … `Len(p) + EC(p)`
- Thus we can run A* using those two metrics
  - `A*-Len(N,M)` … outputs the shortest path between nodes N and M
  - `A*-Cost(N,M)` … outputs the least costly path between nodes N and M
- What do `A*-Len(N,M)` and `A*-Cost(N,M)` look like?

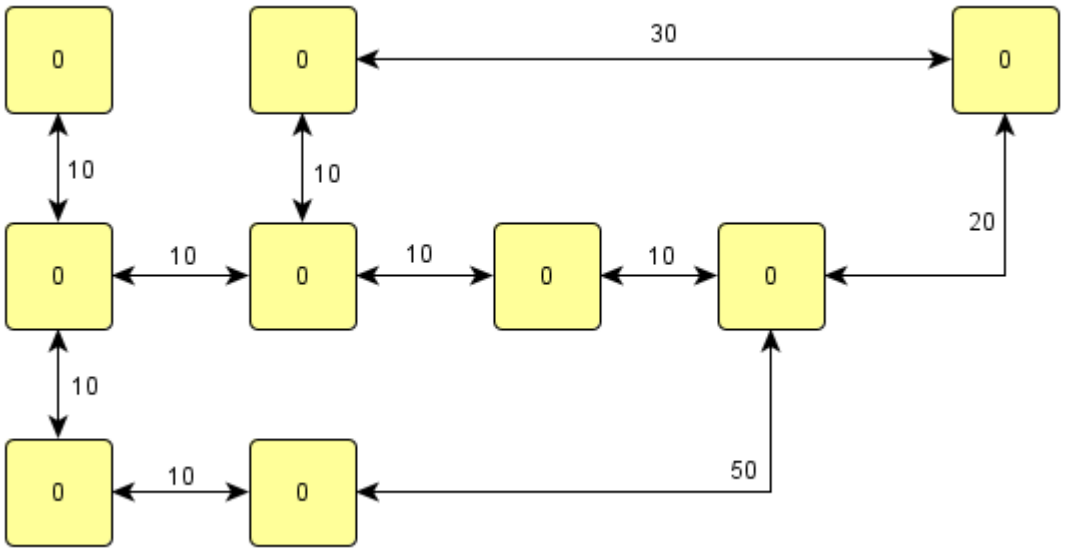
2. `A*-Len(N,M) == A*-Cost(N,M)`
   - Both paths contains `B'` subset of B

⇒ There is no other `PATH(N,M)`, for which following would hold:
- `Cost(PATH(N,M)) < Len(A*-Len(N,M)) + EC(A*-Len(N,M))`
- `Len(PATH(N,M)) + EC(PATH(N,M)) < Len(A*-Len(N,M)) + EC(B')`

⇒ All other paths that would go around `B'` are longer than `EC(B')`!

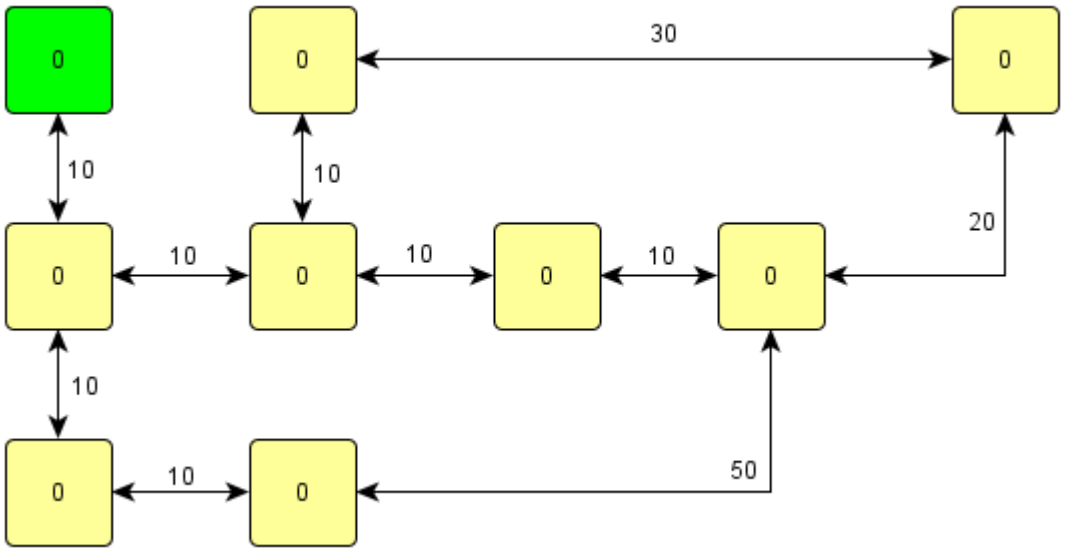
A* Algorithm

Juggling with node/link costs



A* Algorithm

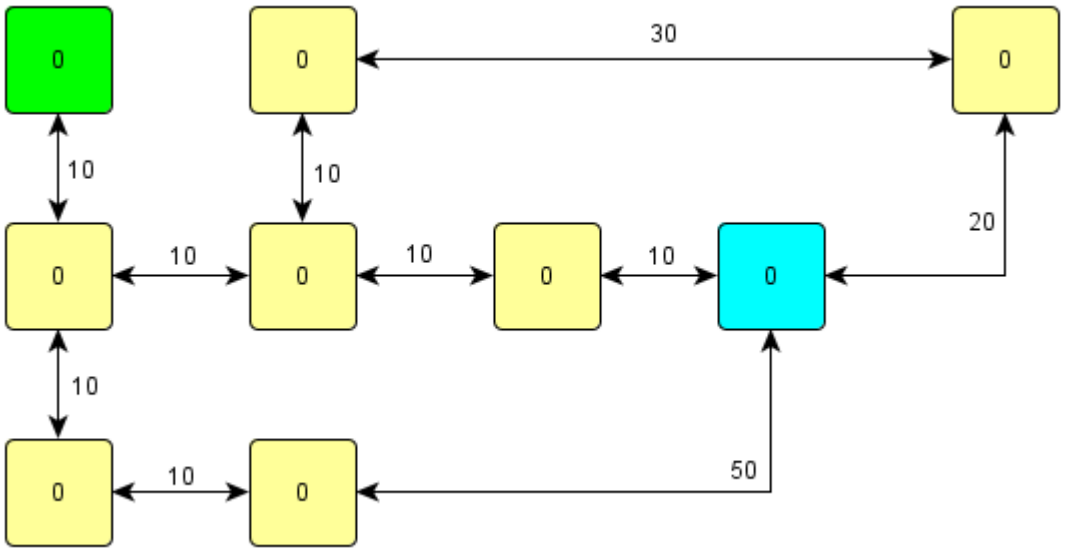
Juggling with node/link costs



- Start-node

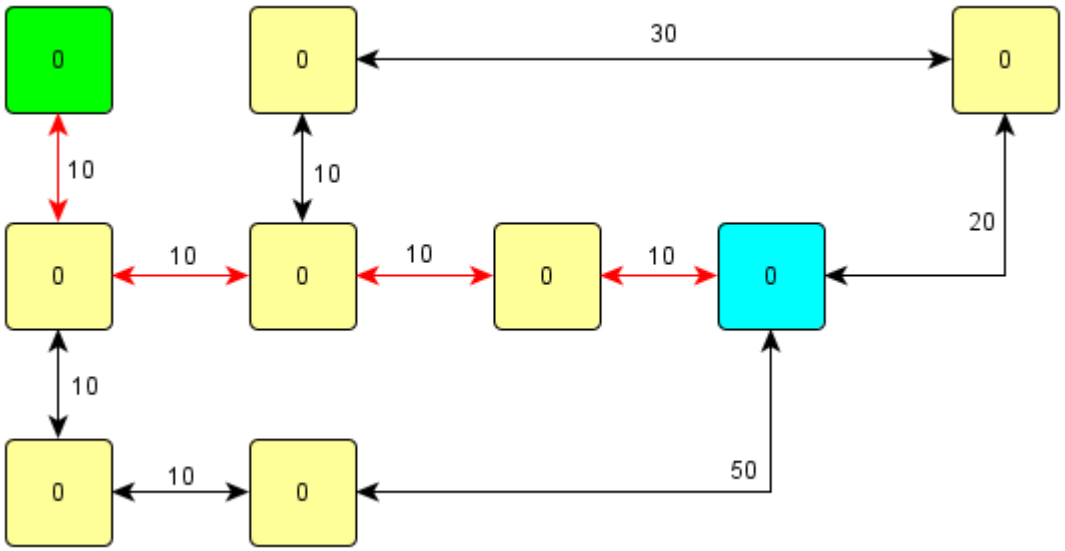
A* Algorithm

Juggling with node/link costs



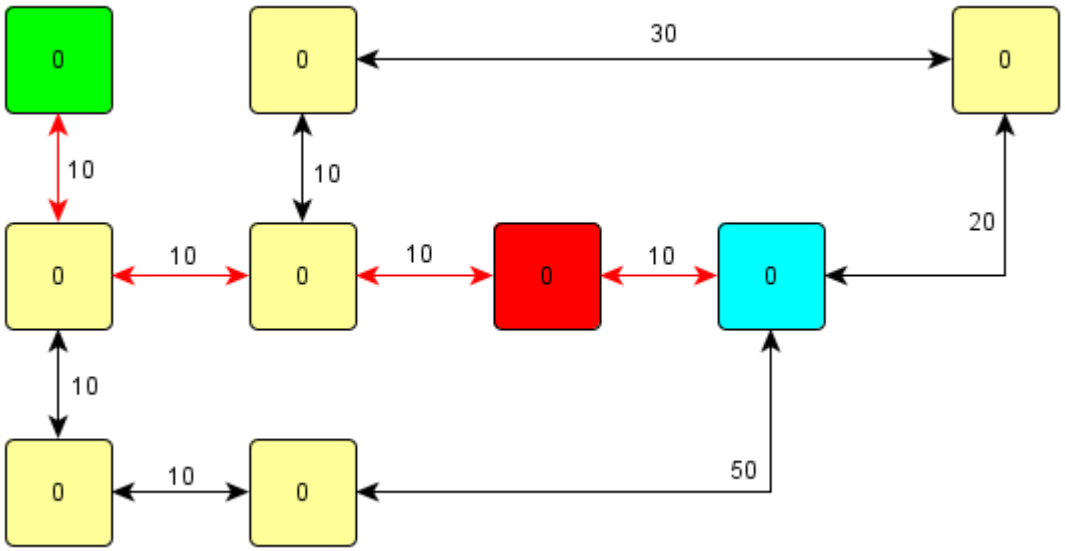
A* Algorithm

Juggling with node/link costs



A* Algorithm

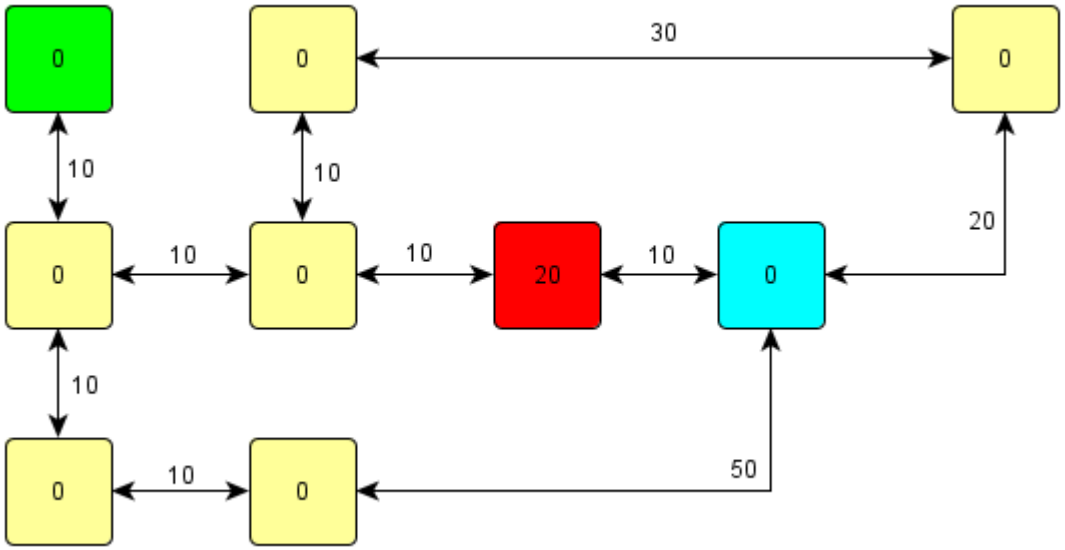
Juggling with node/link costs



A* Algorithm

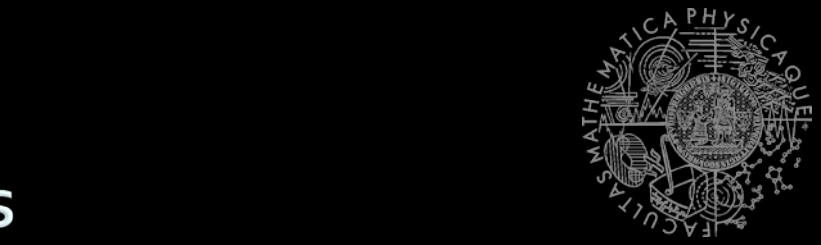
Juggling with node/link costs

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

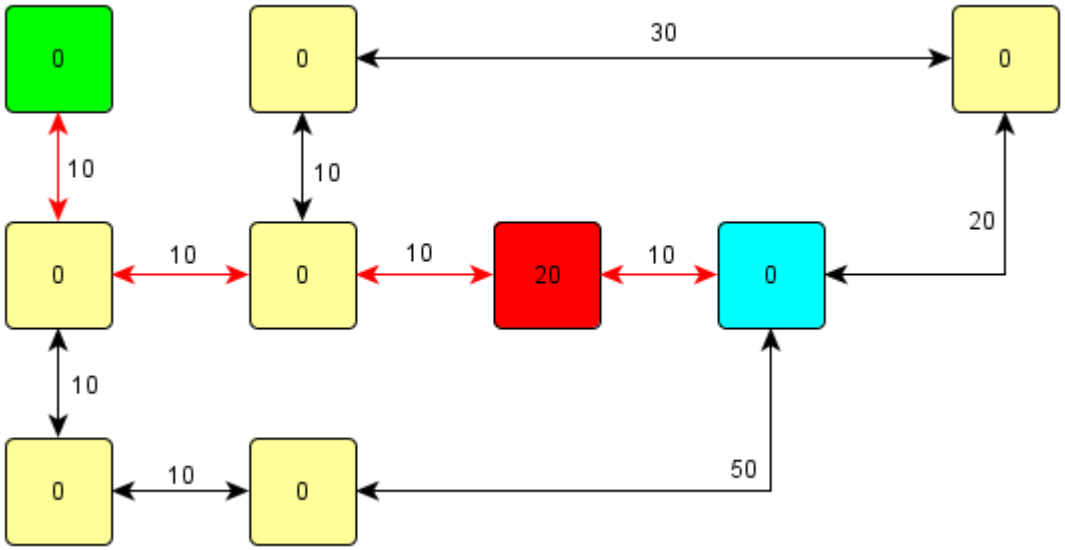


A* Algorithm

Juggling with node/link costs



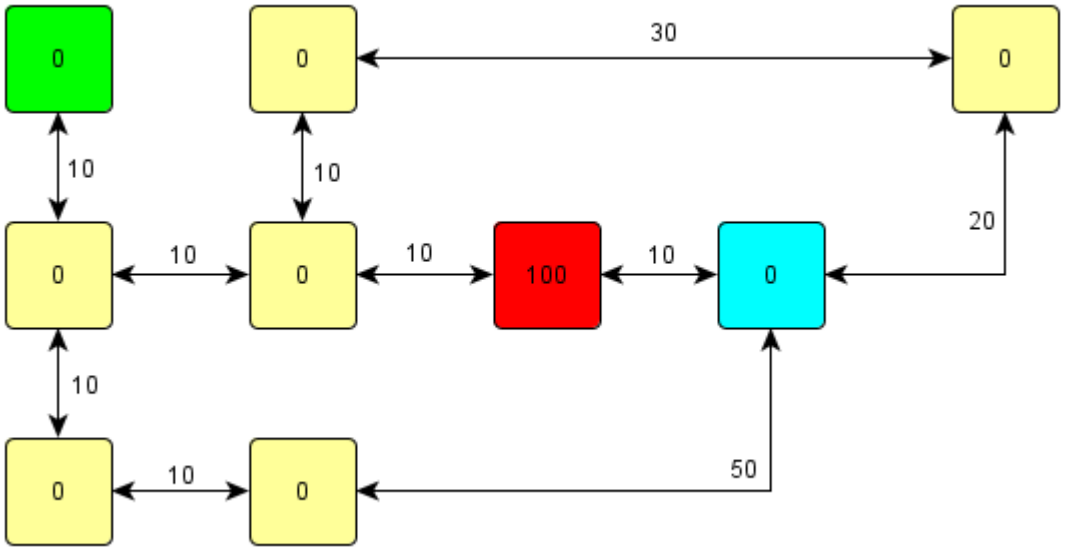
- No…



A* Algorithm

Juggling with node/link costs

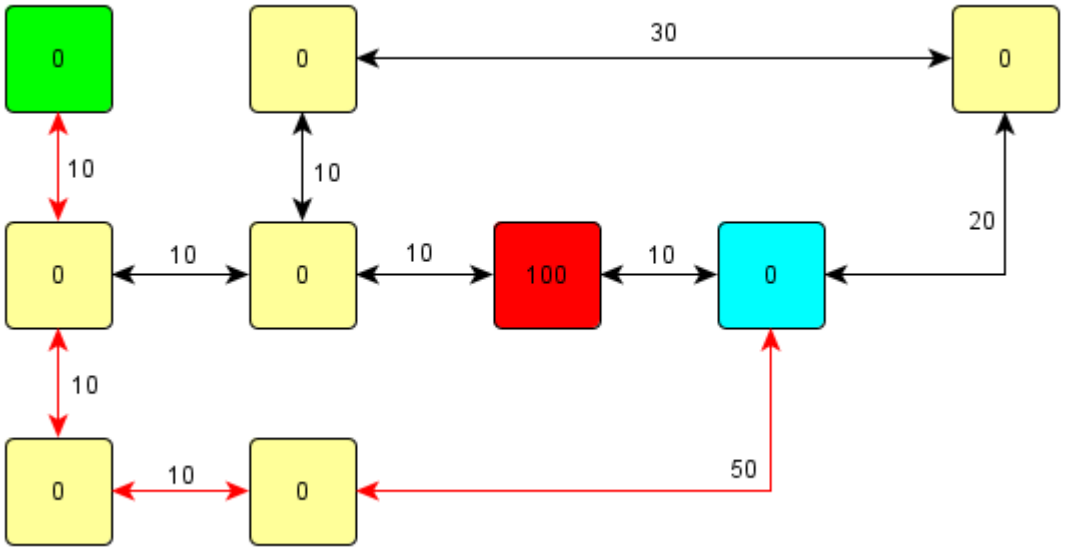
- Rise the NODE cost again… enough now?



A* Algorithm

Juggling with node/link costs

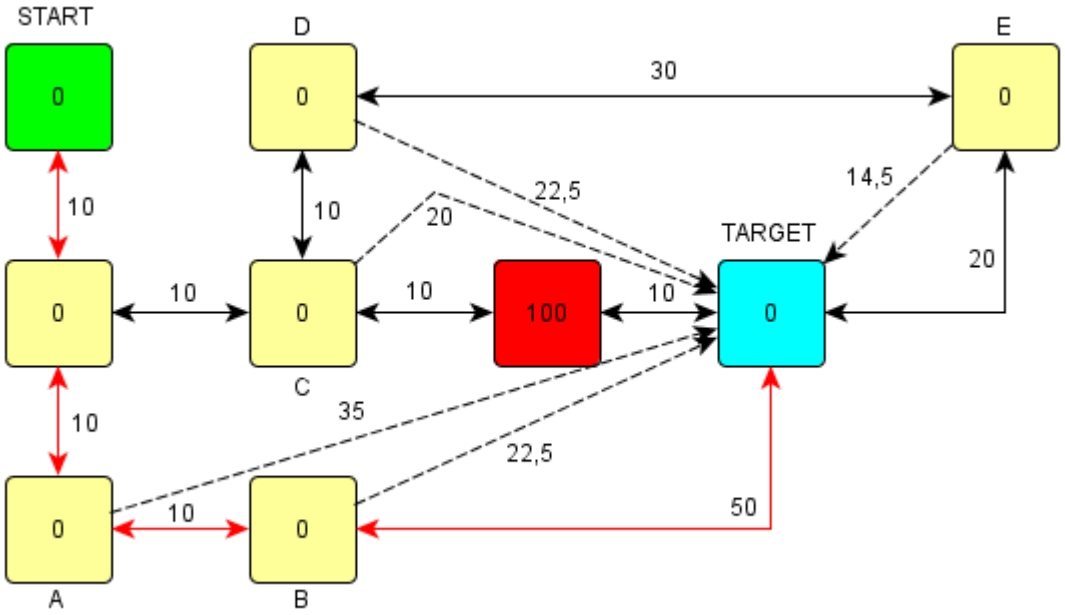
- Here you go!
  - Why was this path found?



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 = 4 games of 10 rounds each of hide and seek with fixed seeker for each game
  - 1 round = 90 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)

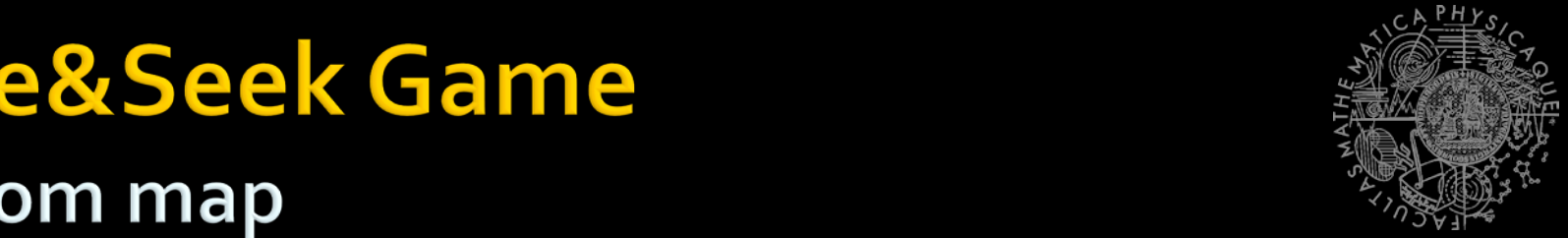
Hide&Seek Game

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**
  - Seeker fouled (dwelled in restricted area > 7 secs) **-100**

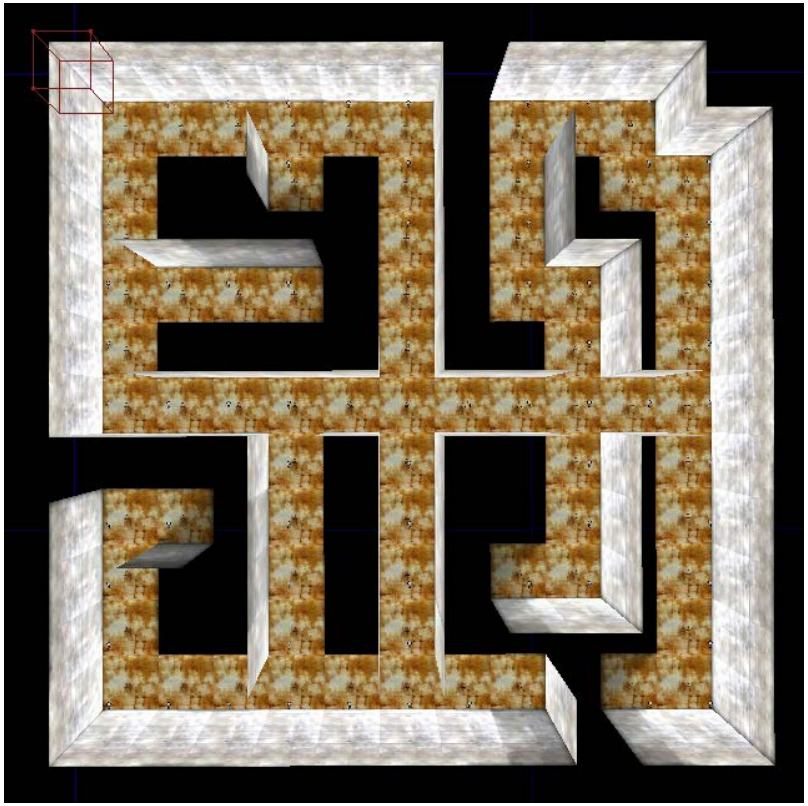
Hide&Seek Game

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!

- 4 bots
  - 1 Seeker, 3 Runners
- Random groups, Fixed map
- Fixed Seeker - 4 matches per group
- Only bots submitted until Sunday 10.4.2016, 8:00 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 2016 – 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
    - Jakub Gemrot (Tuesday 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…

- In case of doubts about the assignment, tournament or hard problems, bugs don't hesitate to contact us!
  - Jakub Gemrot (Tuesday labs)
    - jakub.gemrot@gmail.com