HALMSTAD UNIVERSITY

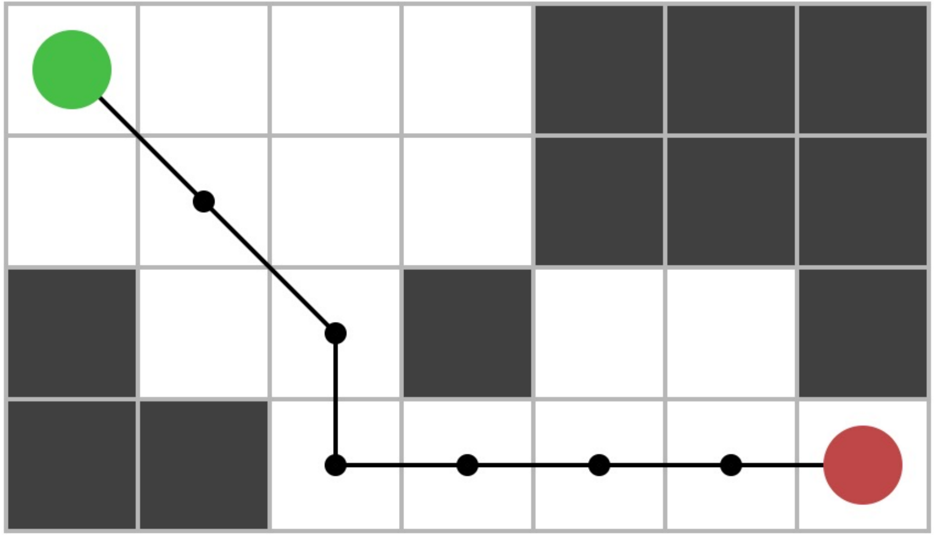
# Artificial Intelligence Laboratory 2: Search Algorithms

DT8042 (HT2021) Halmstad University

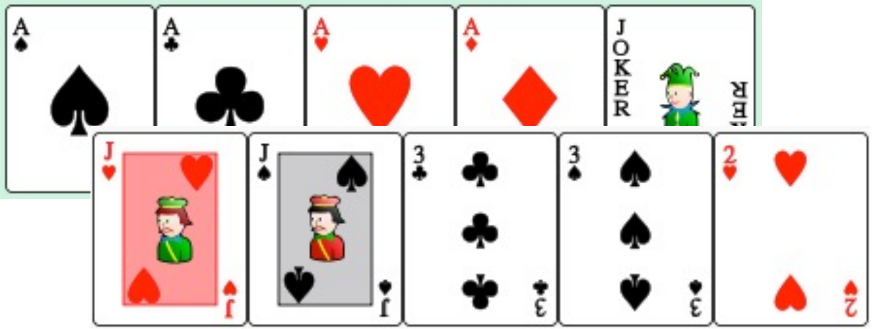
Nov 2021

# Lab 2

- Path Planning
  - – Find a shortest path



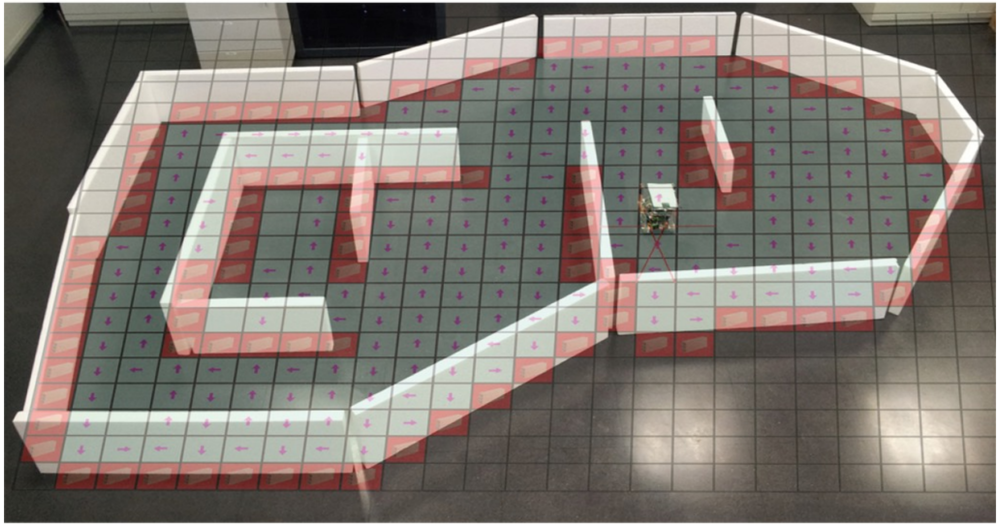
- Simplified Poker game
  - – Find optimal sequence of actions

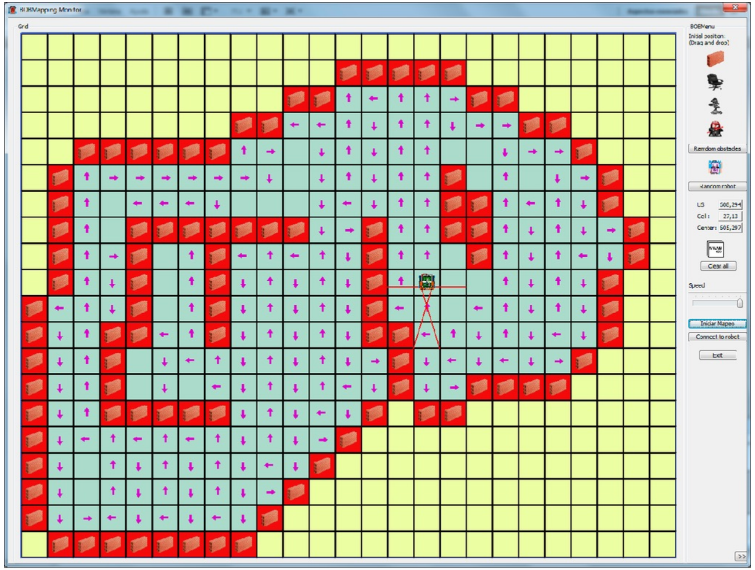


# General Objective

- Implement search algorithms
  - Random search
  - Exhaustive search (BFS & DFS)
  - Greedy search
  - A* algorithm
- Design and investigate different heuristic functions for informed search algorithms
  - reduce the search space
  - using information available

# Occupancy Grid Map

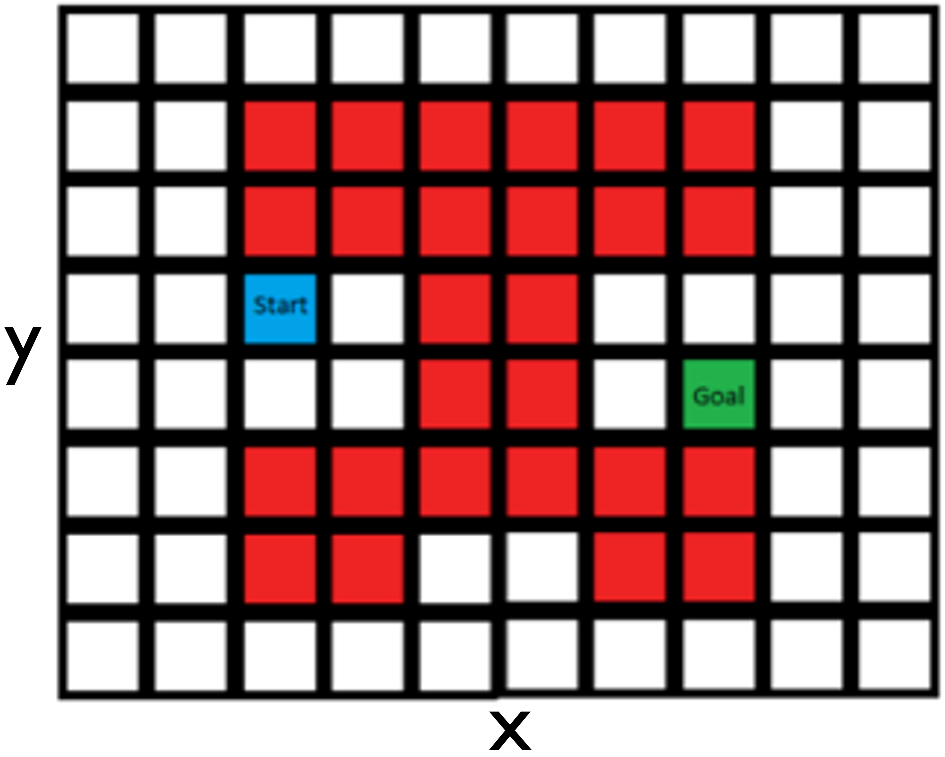




US 506,294
Cell : 27,13
Center: 505,297

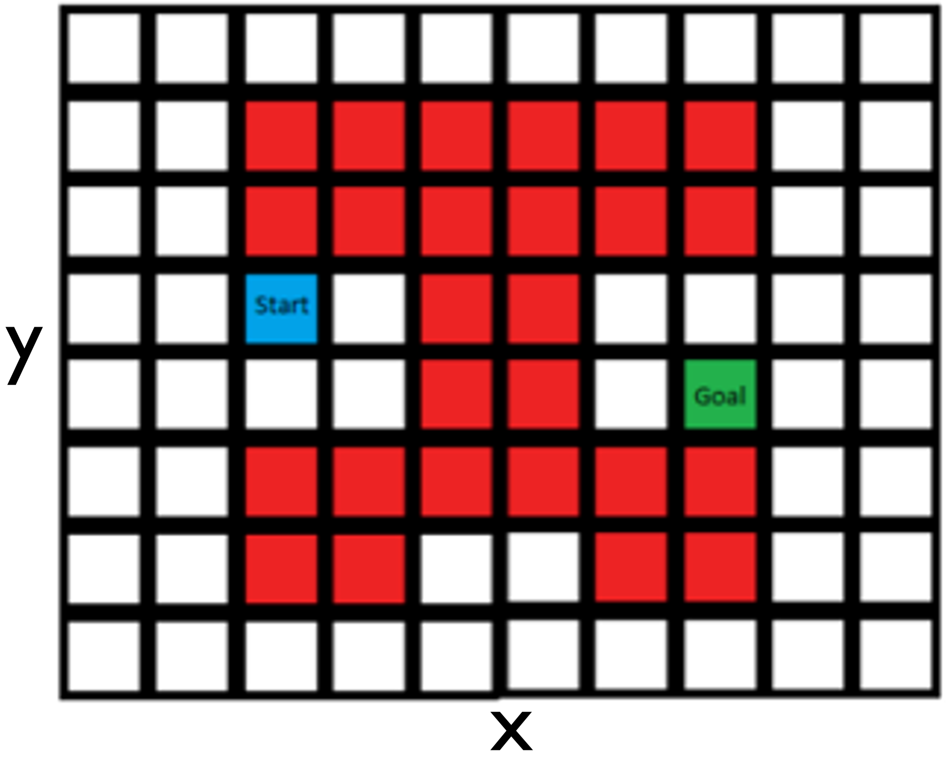
Gonzalez-Arjona, David, et al. "Simplified occupancy grid indoor mapping optimized for low-cost robots." *ISPRS International Journal of Geo-Information* 2.4 (2013): 959-977.

# Path Planning

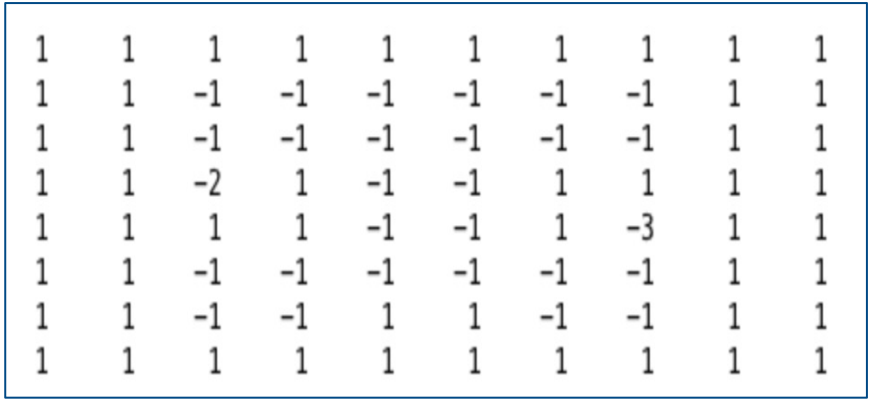


- Red block: obstacles
- Starting point
- Goal

# Path Planning

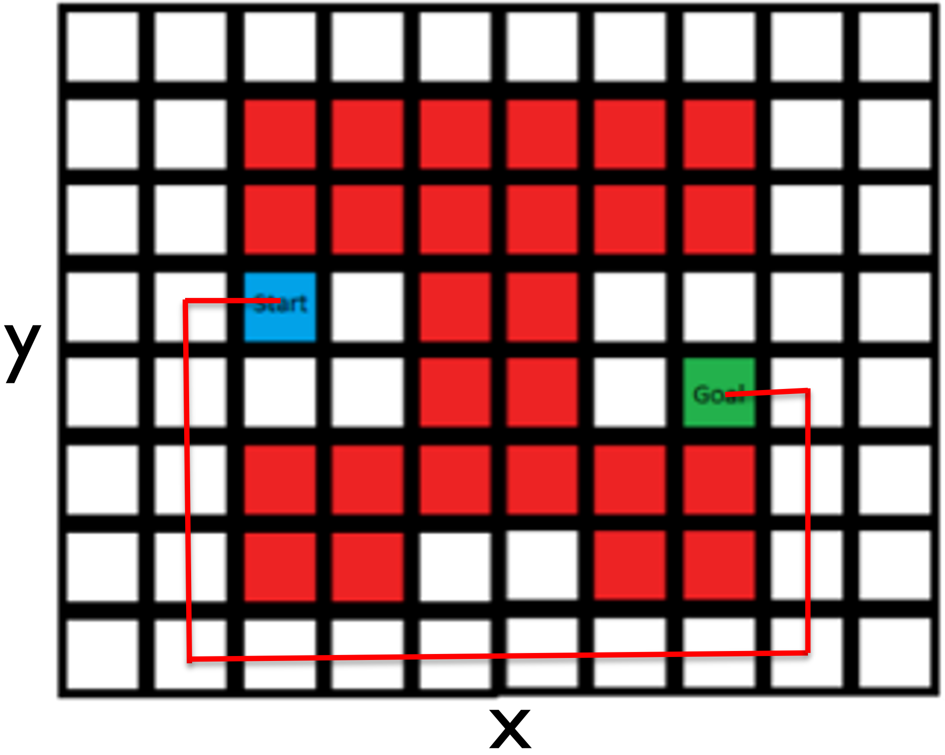


- Red block: obstacles
- Starting point
- Goal



```
1   1   1   1   1   1   1   1   1   1
1   1  -1  -1  -1  -1  -1  -1   1   1
1   1  -1  -1  -1  -1  -1  -1   1   1
1   1  -2   1  -1  -1   1   1   1   1
1   1   1   1  -1  -1   1  -3   1   1
1   1  -1  -1  -1  -1  -1  -1   1   1
1   1  -1  -1   1   1  -1  -1   1   1
1   1   1   1   1   1   1   1   1   1
```

# Path Planning



- Red block: obstacles
- Starting point
- Goal

- Find short path

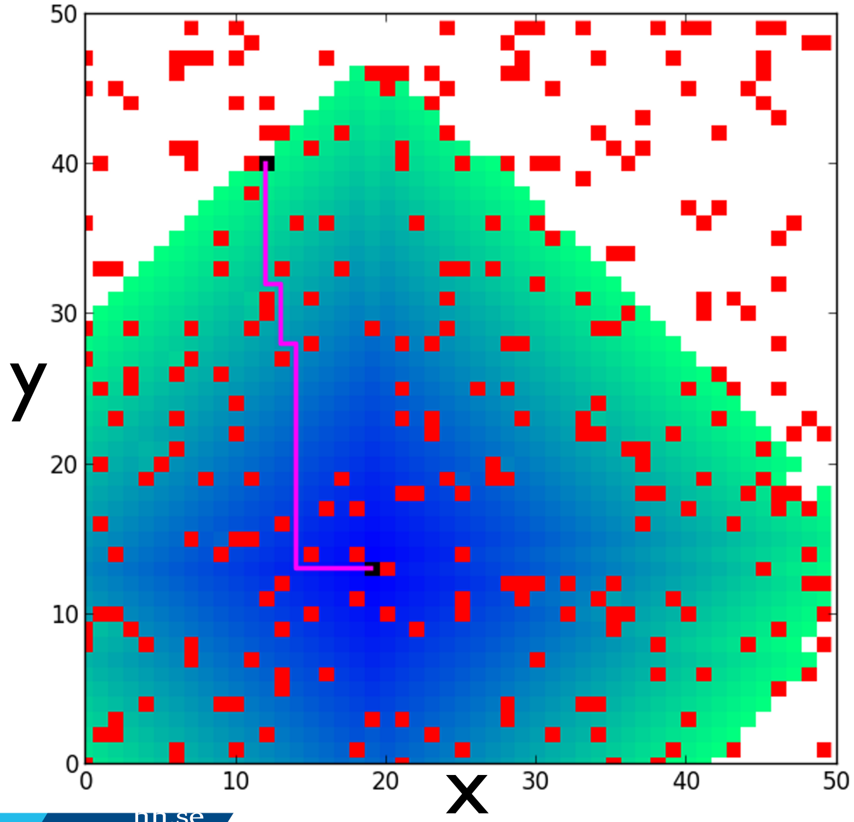
# Grid map

- Starting point - L
- Goal – I
- Queue = {}

- S1 - open (L, 0)
  - Queue = {(G, 1), (K, 1), (Q, 1)}
- S2 - open (G, 1)
  - Queue = { (K, 1), (Q, 1), (B, 2), (F, 2)}
- S3 - open (K, 1)
  - Queue = { (Q, 1), (B, 2), (F, 2), (P, 2)}
- S4 – open (Q, 1)
  - Queue = {(B, 2), (F, 2), (P, 2)}
- S5 – open (B, 2)
  - Queue = {(F, 2), (P, 2), (A, 3), (C, 3)}
- S6 and 7 – open (F, 2), (P, 2)
  - Queue = {(A, 3), (C, 3)}
- S8 – open (A, 3)
  - Queue = { (C, 3)}
- S9 – open (C, 3)
  - Queue = { (D, 4)}
- S10 – open (D, 4)
  - Queue = {(I, 5), {E, 5}}

| | | | | |
|---|---|---|---|---|
| A, 3 | B, 2 | C, 3 | D, 4 | E, 5 |
| F, 2 | G, 1 | H<br>WALL | I, 5<br>END | J |
| K, 1 | L<br>START | M<br>WALL | N | O |
| P, 2 | Q, 1 | R<br>WALL | S | T |

# Path Planning



Red block: obstacles

Starting point

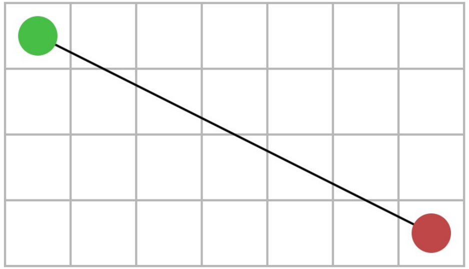
Goal

Find short path

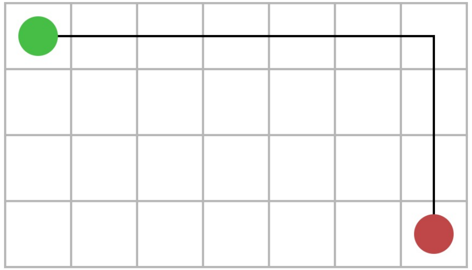
– Reduce search space!

# Heuristic function

$$h(n)^2 = (n.x - goal.x)^2 + (n.y - goal.y)^2$$



$$h(n) = |n.x - goal.x| + |n.y - goal.y|$$



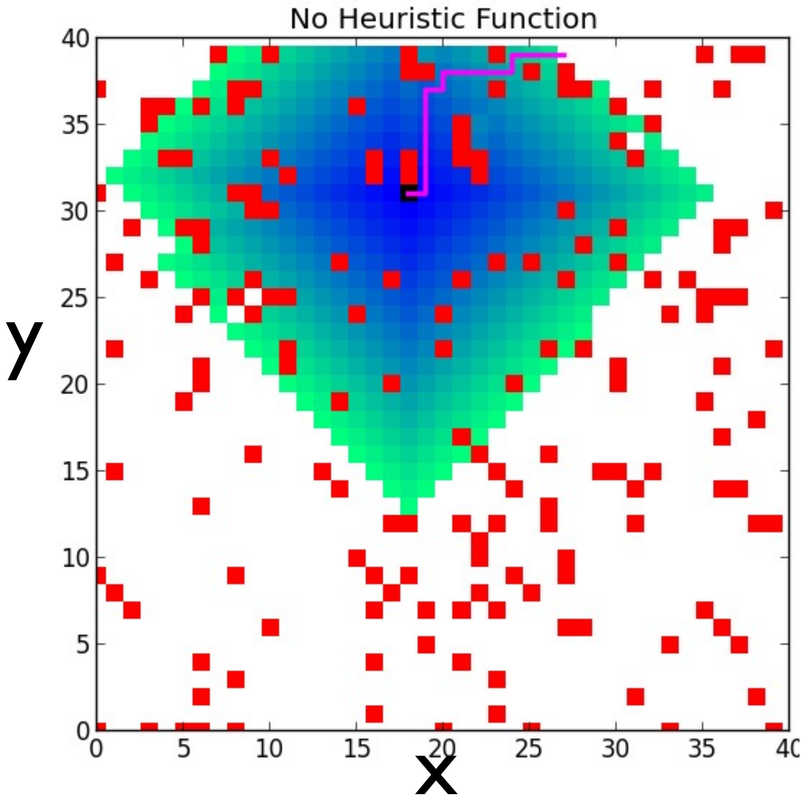
General-purpose heuristics for 2d grid map

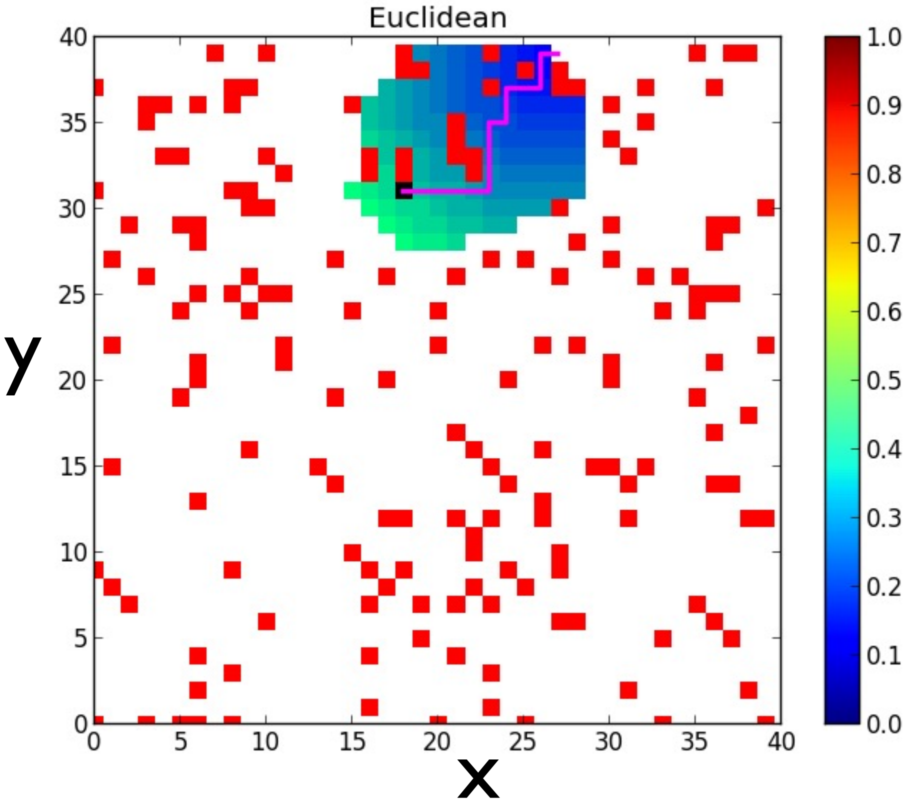
– Euclidean distance

– Manhattan distance

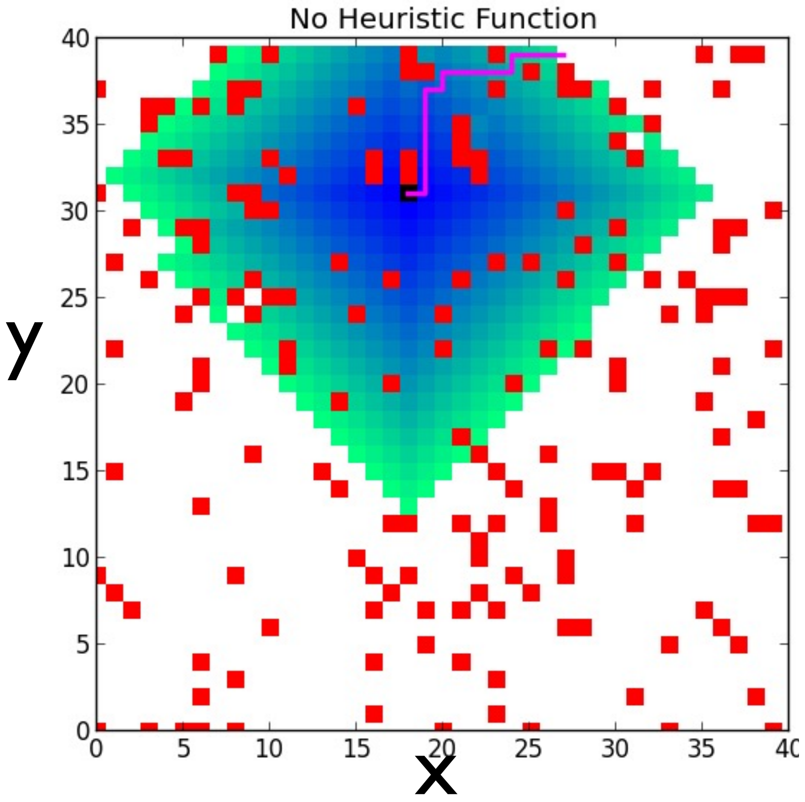
# Breadth-first search

# A* with Manhattan distance as heuristic

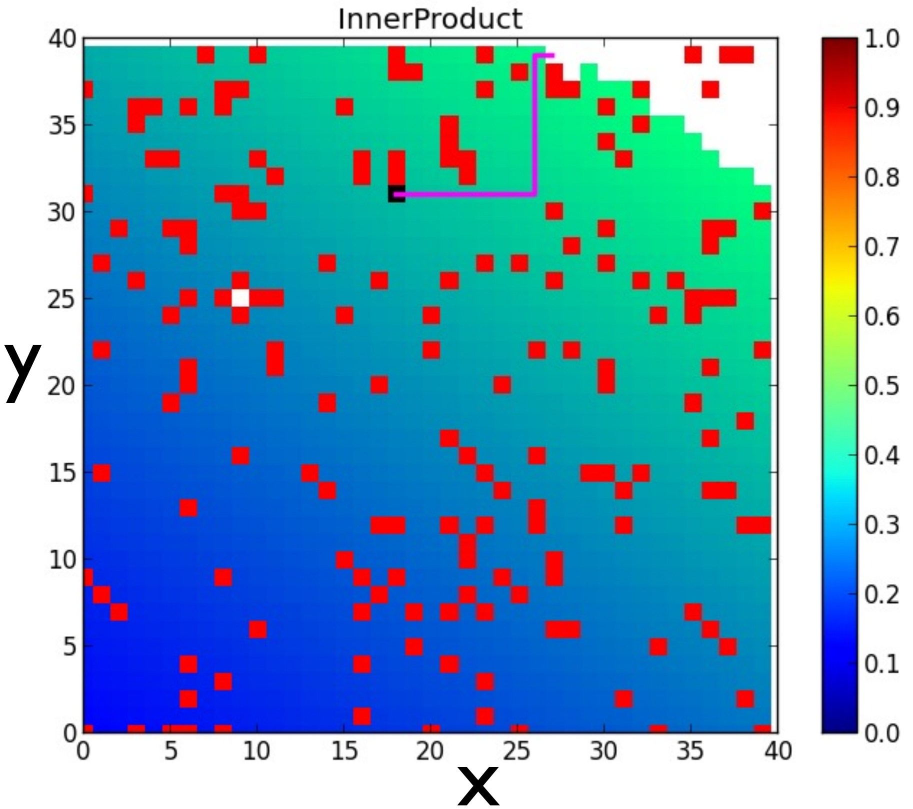




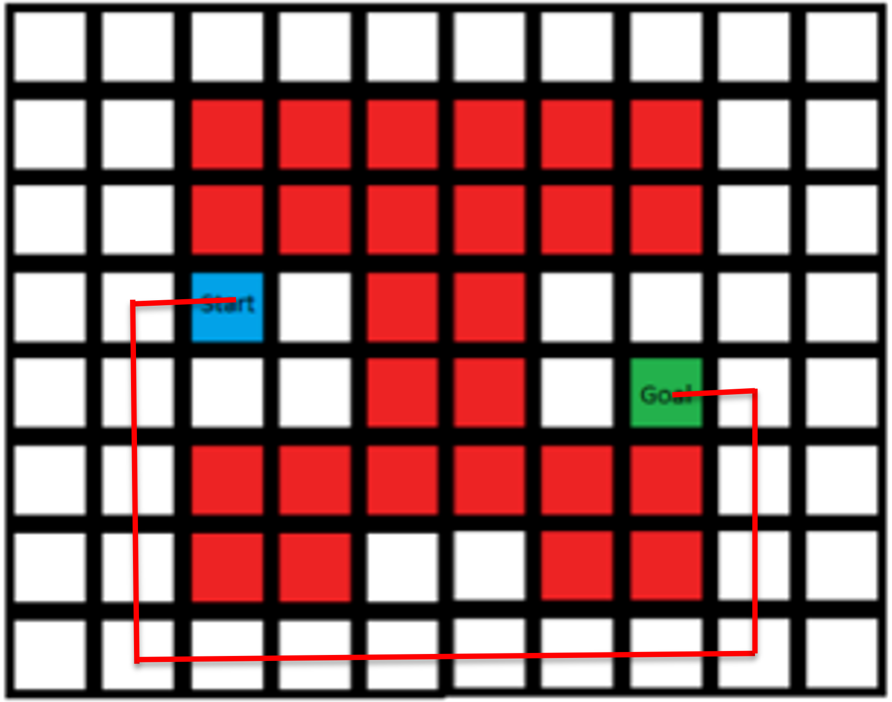
# Breadth-first search



# A* with Inner Product distance as heuristic

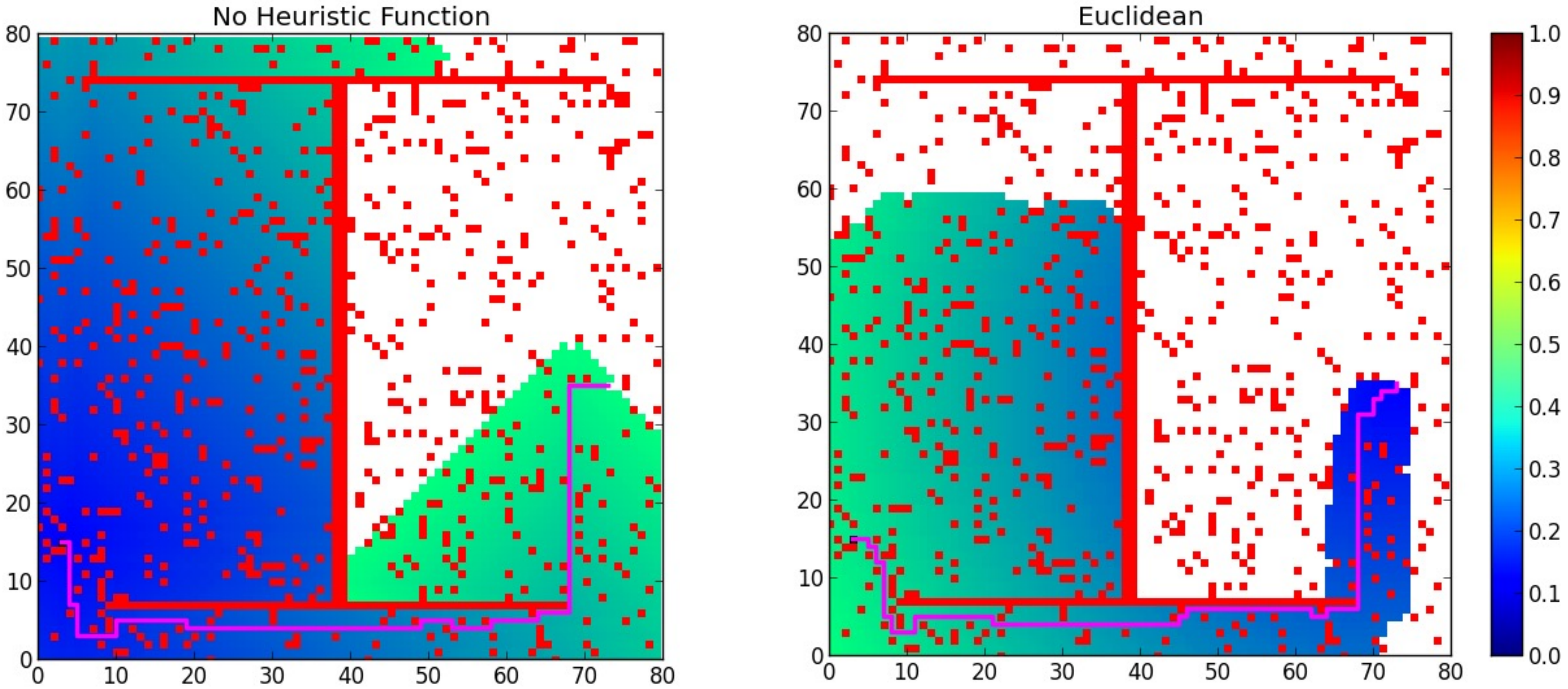


# Environment-specific Heuristic



- Env. Information
  - Starting point at left side and ending point at right side
  - The environment contains a ‘工’ shaped obstacle
    - Y coordinate of upper and lower edges of ‘工’
    - X coordinate of the vertical wall

# Path Planning in ‘工’ environment



# Task 1

- Uninformed/Informed search algorithms
  - A* Star algorithm - try different types of heuristics
  - Check whether the solution is optimal
  - Total number of nodes opened
- 2 types of environment
  - Map with obstacles randomly placed
  - Map with ‘工’ obstacles and randomly placed obstacles
- Comparison with random search, exhaustive search and greedy search
  - Modify cost function
- Report
  - Plots of solved maps (two types environment) with path and value of evaluation function for each cell encoded in visual features
  - Comparison result shall be based on multiple instances

# Practical Details

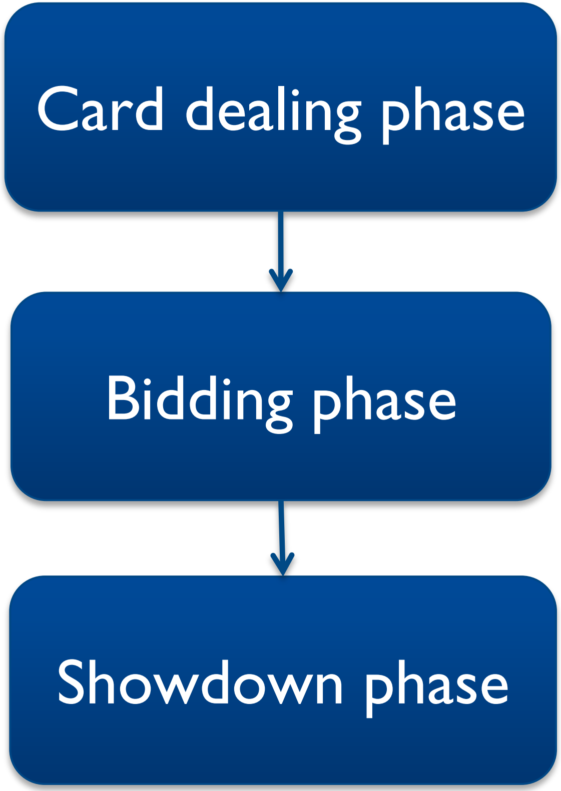
- If you have any problem implementing the search algorithm, please go through the following tutorial
  - http://swarm.cs.pub.ro/~anpetre/dynamic_prog.pdf
- Please feel free to work with your coding preference
  - Data structure (e.g. Classes/objects, list etc. for cells in task1)
  - Structure of the search algorithm

# Task 2 Poker game

- Rules: slightly more complex than the first lab!
- Search optimal solution
  - Random, exhaustive and Greedy search
  - A algorithm with heuristic
- Objective
  - Design a special heuristic function that reduces the search space

# Game flow (1$^{st}$ lab)

- Card dealing phase
  - Assign 3 cards to agents
- Bidding phase
  - Amount $0-50
  - Regardless of how much other players bet
- Showdown phase

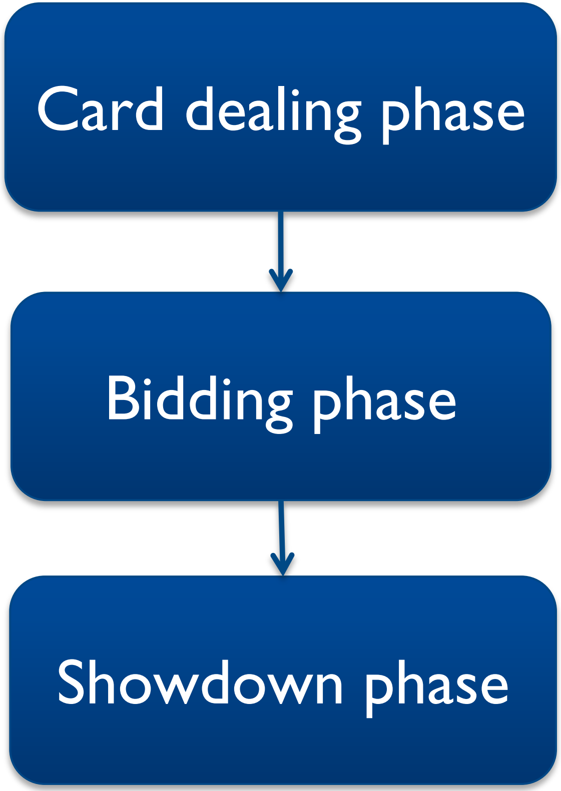

Card dealing phase

Bidding phase

Showdown phase

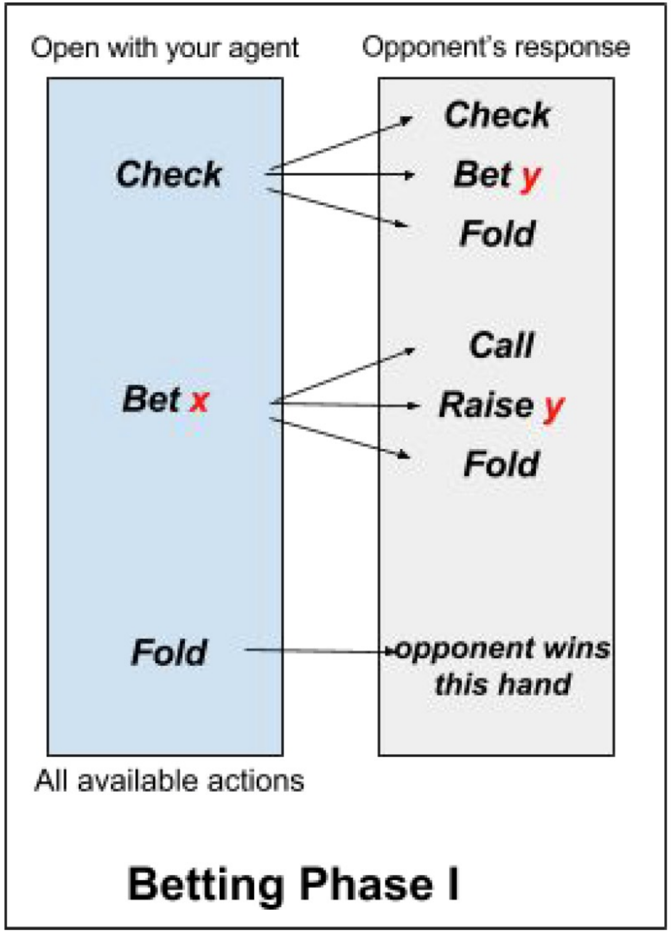
# Game flow

- Card dealing phase
  - Assign 5 cards to players
- Bidding phase
  - Search sequence of actions
- Showdown phase



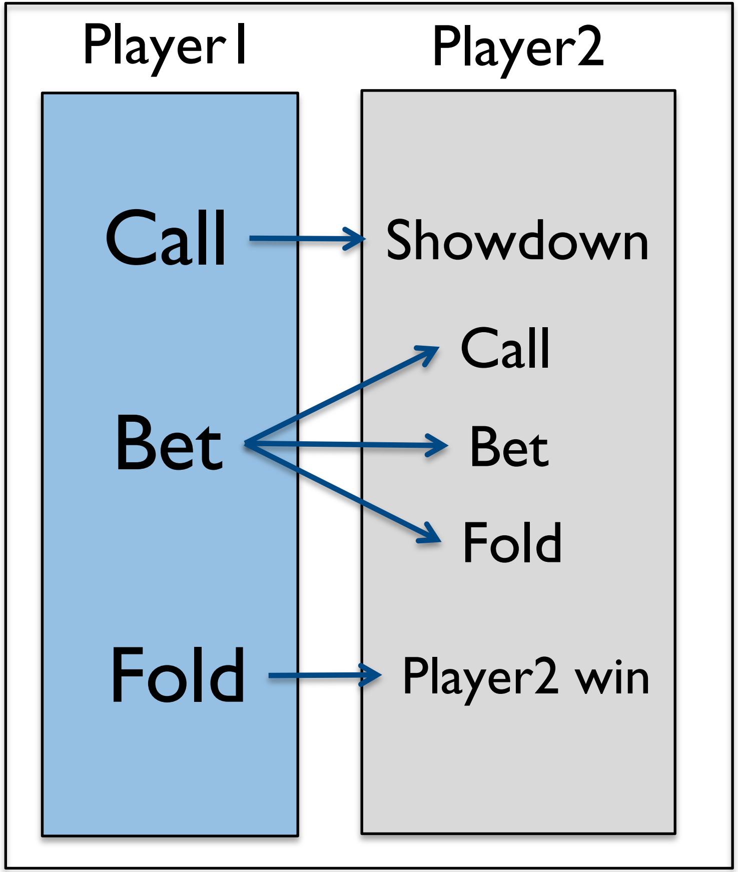
# Traditional Poker game

- Actions Available
  - Bet
  - Raise
  - All In
  - Check
  - Call
  - Fold
  - Showdown



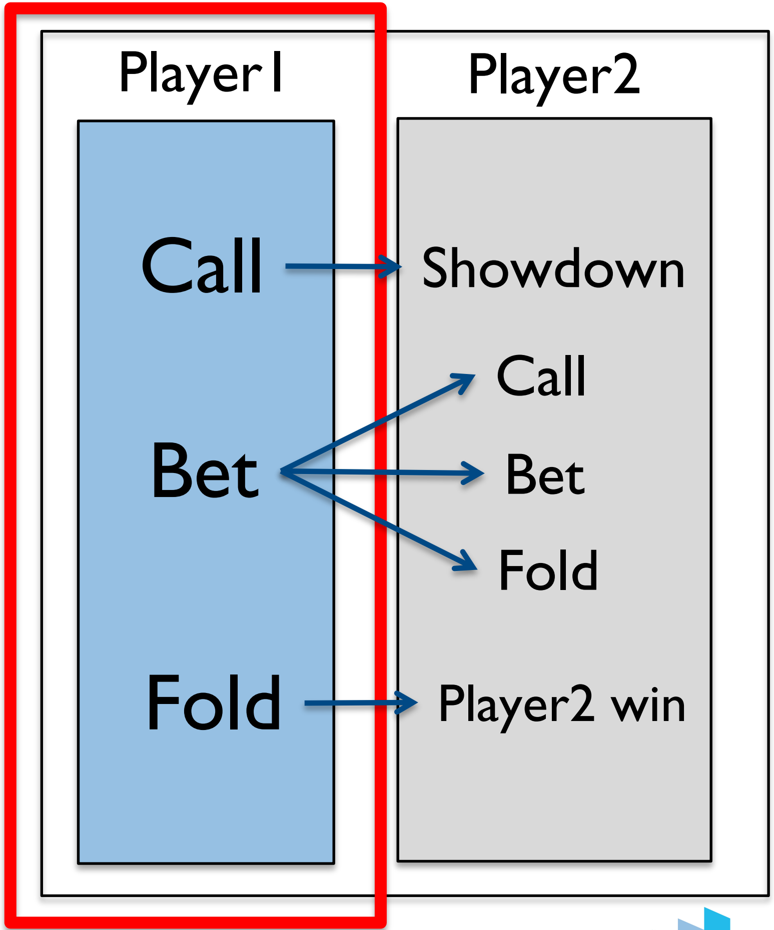
# Simple Poker Game

- Action Available
  - Bet x coins
    - 5, 10 or 25 coins
    - Regardless of how much opponent bet
  - Call
    - Putting 5 coins in the pot and show hand
  - Fold



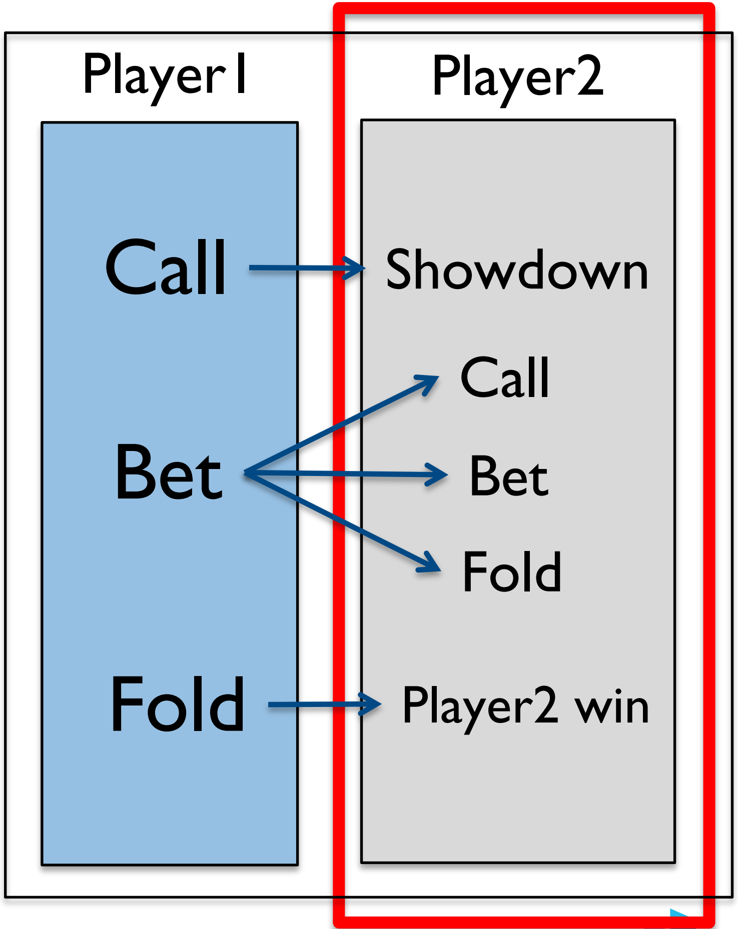
# Simple Poker Game

- Always start with your agent
- Action Available
  - Bet x coins
  - Call
  - Fold



# Simple Poker Game

- Always start with your agent
- Action Available
  - – Bet x coins
  - – Call
  - – Fold



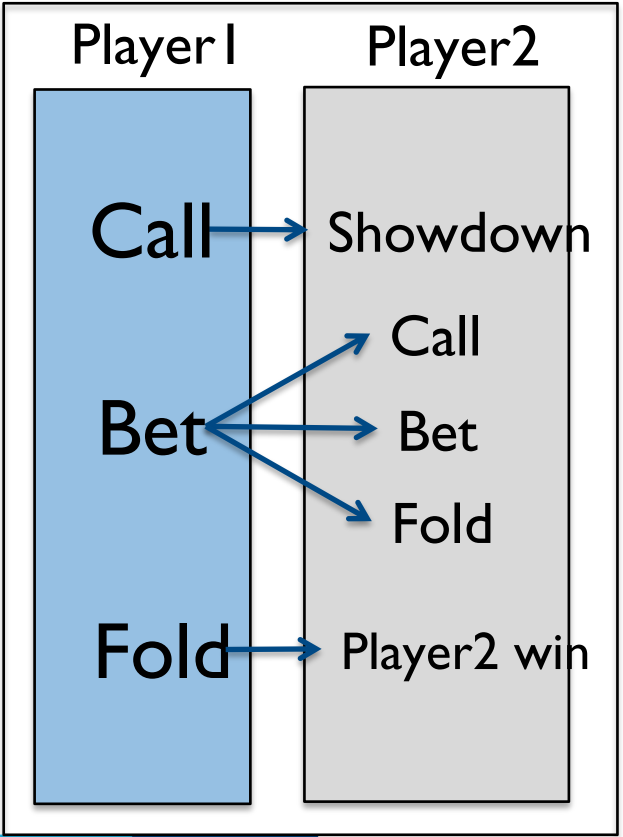
# Tasks

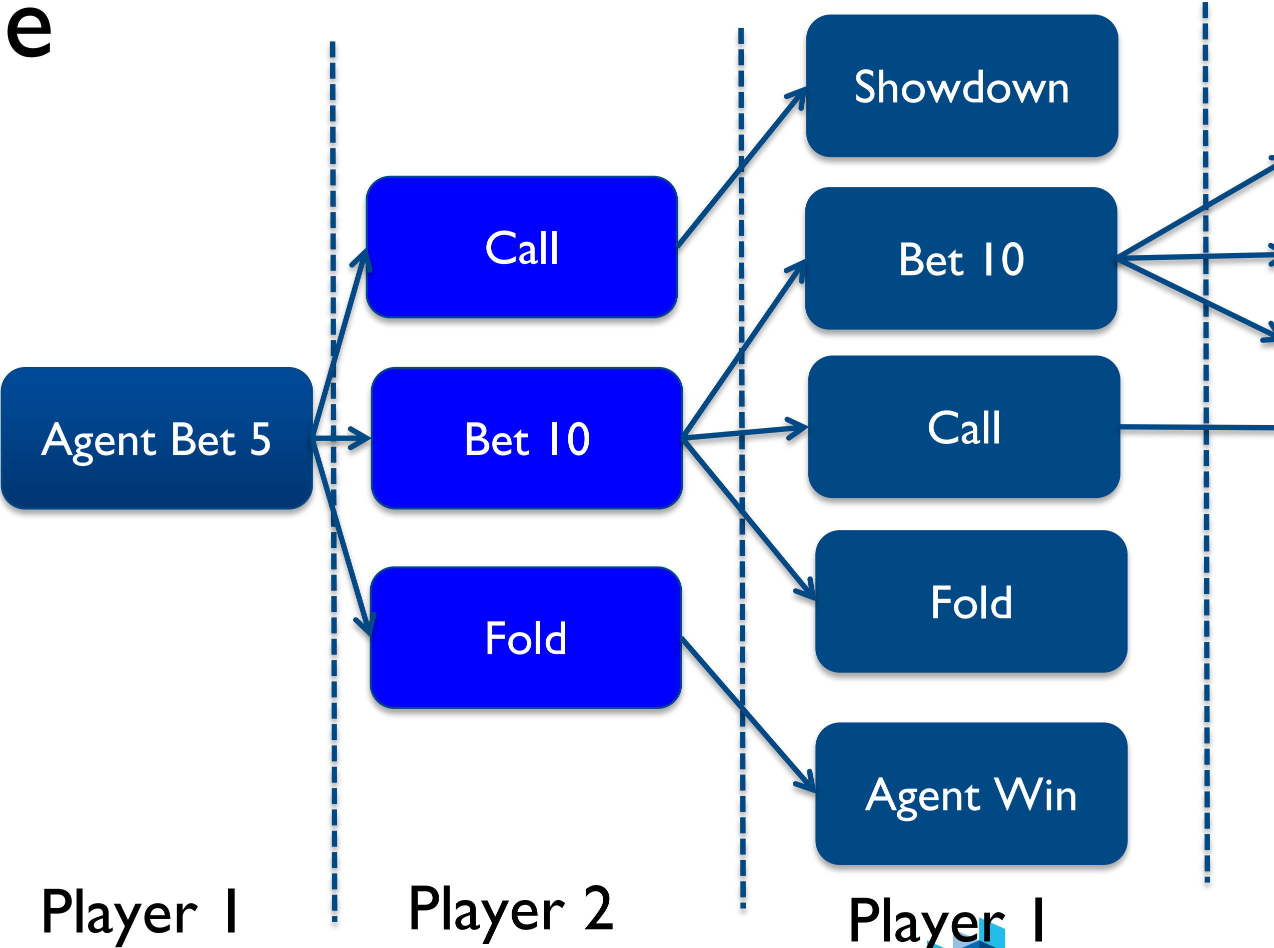
- Build environment of the game
  - Hand evaluation function for 5 cards
  - Function for updating the state of the game
    - Number of hands played, coins left for both agent, coins in the pot, current hand for both agent etc.
- Implement two fixed agents playing against each other

# Expectation / Tasks

- Use search algorithm to find a series of actions for your agent to win more than 100 coins within 4 hands
  - given known strategy of the opponent and complete information of the game
  - Start with exhaustive search
- Design heuristic function that reduce the search space

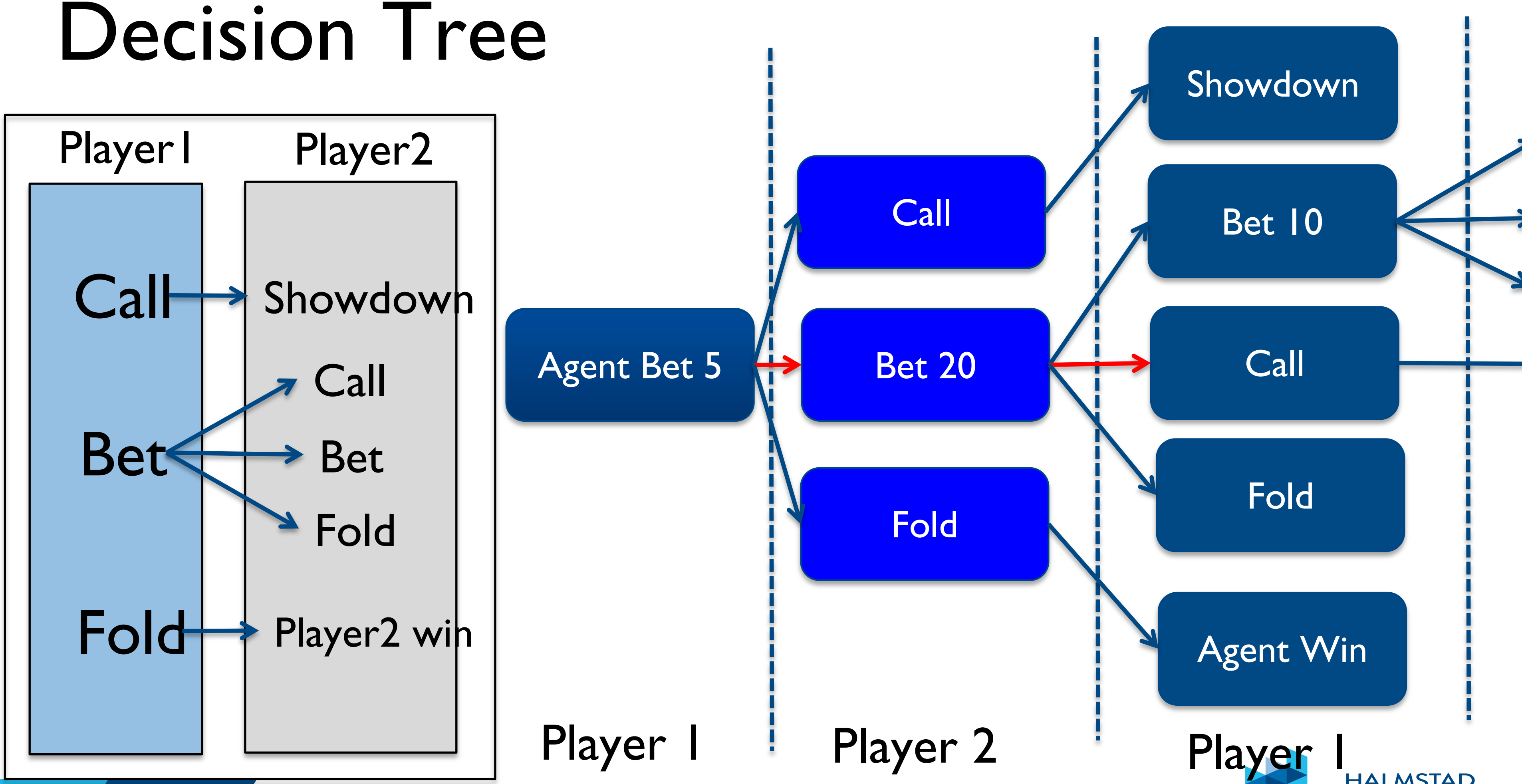
# Decision Tree



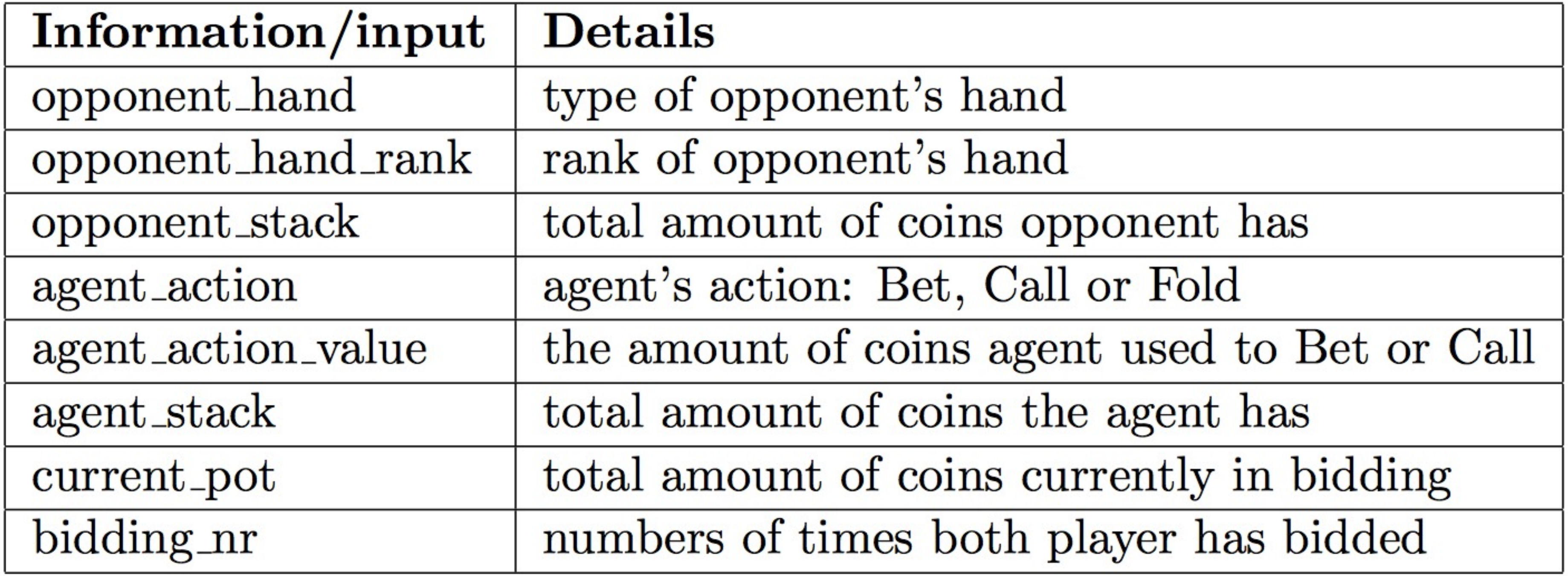


HALMSTAD UNIVERSITY

# Decision Tree

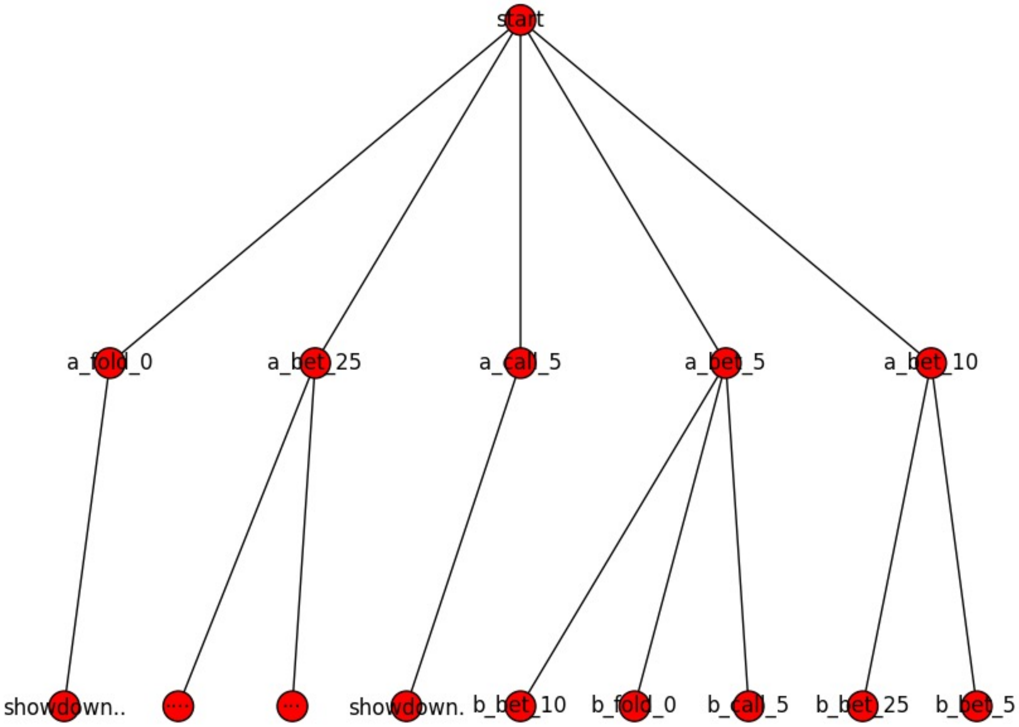


# poker_strategy_example(...)



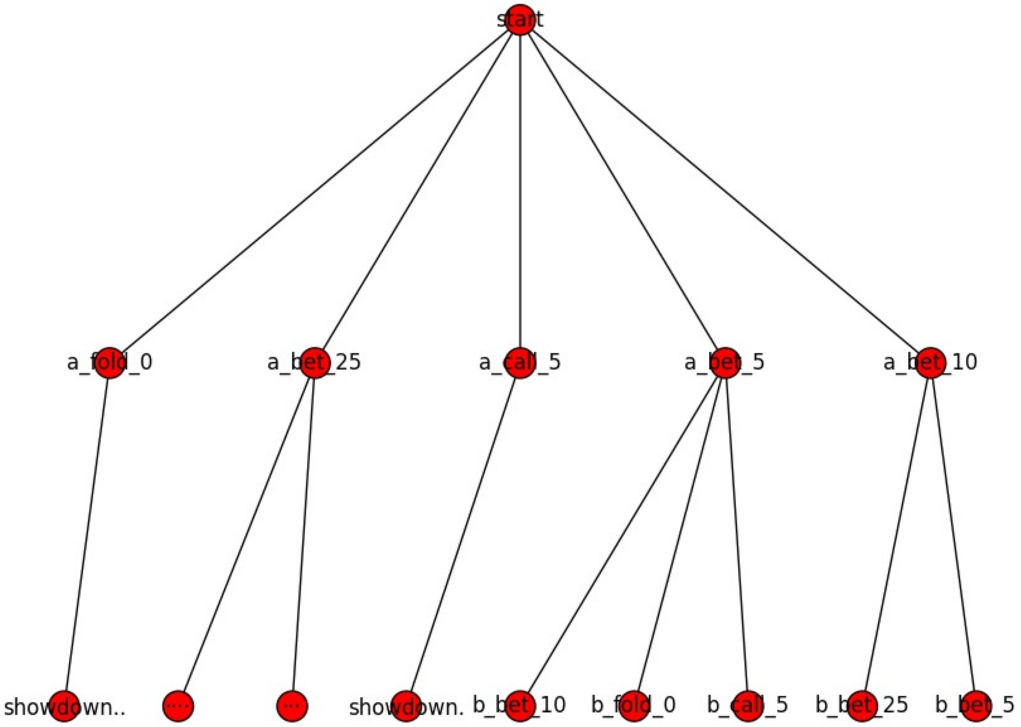
| Information/input | Details |
|---|---|
| opponent_hand | type of opponent's hand |
| opponent_hand_rank | rank of opponent's hand |
| opponent_stack | total amount of coins opponent has |
| agent_action | agent's action: Bet, Call or Fold |
| agent_action_value | the amount of coins agent used to Bet or Call |
| agent_stack | total amount of coins the agent has |
| current_pot | total amount of coins currently in bidding |
| bidding_nr | numbers of times both player has bidded |

# Start by generating tree for one hand



- Generate a tree
- Give each node an identifier
  - Depth
  - Round
  - Agent actions
  - …
- Generate all edges

# Start by generating tree for one hand



- State
  - Depth
    - Hand/round nr.
    - Bidding nr.
  - Current pot
  - Agent
    - Hand
    - Stack
    - Action, value
    - Coins bidded in current hand
  - Cost/heuristics
  - …

# Expectation / Tasks

- Implement environment of the game
  - Evaluation function
  - Update state of the game
  - 2 fixed agent playing against each other
- Implement and apply random search, exhaustive search and greedy search to find shortest sequence of actions
- A* with customized heuristic function (optional)

# Grading

- Pass/fail/extra credits
- Submit your lab on the Blackboard
  - a report about what you have done
  - Code