project 2:

game trees and path planning

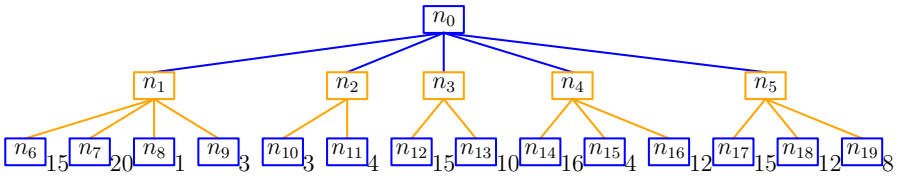
solution(s) due:

June 20, 2016 at **12:00** via email to **bauckhag@bit.uni-bonn.de**

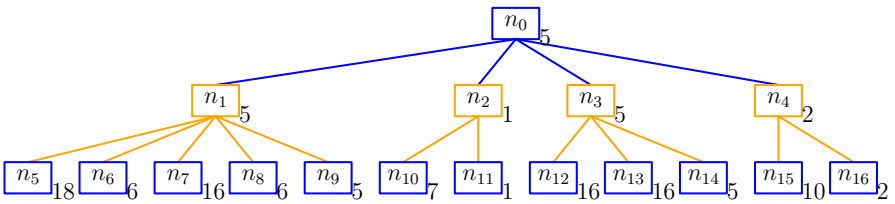
problem specification:

task 2.1: the *tic tact toe* game tree: in the lecture, we (over)estimated the number of *tic tac toe* game states to be $3^9 = 19.683$.
Now, using combinatorial arguments, compute an upper bound for the number of nodes in the complete *tic tact toe* game tree (starting with an empty board and player **X** to move). This number should be considerably larger than the number of game states; explain how this is possible.
Next, compute the complete *tic tac toe* game tree (starting with an empty board and player **X** to move and ignoring symmetries). This may take a while! How many nodes does the tree really have? How many of these nodes correspond to a situation where player **X** wins? What is the average branching factor of the tree?

task 2.2: minmax computations: implement the minmax algorithm and use your implementation to compute $mmv(n_0)$ for the following tree



Now, consider the following tree

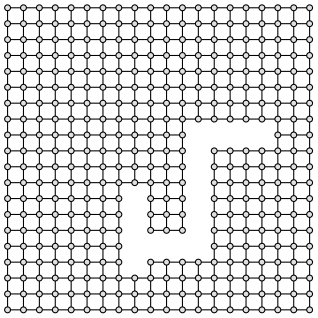


where minmax values have already been computed for all its nodes. Observe that n_1 and n_3 both have a minmax value of 5 but the best possible outcome (from the point of view of MAX) in the subtree below n_1 is 18 whereas in the tree below n_3 it is just 16. This would suggest MAX moves to n_1. How would you have to modify your implementation of the minmax algorithm such that it can keep track of "better alternatives" in case of ties? Would you really have to do this? Explain both your answers.

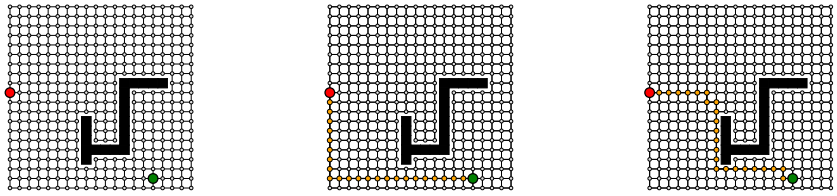
task 2.3: minmax search for *connect four*: extend your code for *connect four* on a 6×7 board from the last project towards more intelligent game play. Think about an appropriate evaluation function that evaluates the merits of non-terminal nodes in a corresponding search tree and use this function to realize a depth-restricted search algorithm. Use this depth-restricted search algorithm to compute the moves of one of the players while the other player still moves at random. Have both players play a tournament of many games and gather the win/loss/draw statistics. What happens if you modify the depth parameter of your depth-restricted search algorithm?
note: in the next project, we will up the ante and consider connect four on larger boards than 6×7. It may thus be a good idea to already think about evaluation functions that do not depend on the fact that the original version of *connect four* is played on a 6×7 board ...

task 2.4: breakout: implement a controller for the breakout game of your choice. That is, write a function that controls how the paddle has to be moved in order to hit the ball. Since this is a rather trivial problem, let's make the overall setting more interesting: modify the code of your breakout implementation such that the speed of the ball increases over time and see how your paddle agent copes with this.

task 2.5: path planning: download the file `simpleMap-1-20x20.txt` from the Google site. It contains a 20×20 matrix of zeros and ones. Think of this matrix as the representation of a 2D *game map* where fields marked 0 are locations a player can be in whereas fields marked 1 represent parts of walls.
Implement a program that reads data like these and transform them into a grid graph (a.k.a. a lattice) where there is a vertex for every zero and an edge between any two vertically or horizontally adjacent zeros. Fields marked 1 should not occur in your graph. As a picture says a thousand words, here is an illustration as to how your graph should look like:



note: make sure your way of reading map matrices and creating graphs therefrom works in general, because shortly before June 20, another file will be uploaded to the Google site for which you will have to show that the code you implement in this task can cope with it.
Assume the $(0, 0)$ coordinate of your game world coincides with the vertex in the lower left corner of the graph. In the following picture, the vertex colored in red is thus located at grid coordinates $(0, 10)$ and the green one resides at $(15, 1)$.



Now, implement algorithms that plan a path from the source vertex s (red) to the target vertex t (green). Proceed as follows:

1. implement Dijkstra's algorithm (for instance, by using `networkx`). The resulting path should resemble the one in the central panel of the above figure. This is a valid path, but it does not look like a path a human player would choose to move from s to t.

2. implement the A^* algorithm to plan a path. To estimate the *future path costs* of a vertex v considered for expansion, use the Euclidean distance $d(v, t)$. Now, your result should more or less look like the one shown in the rightmost panel of the above figure.

general hints and remarks

- Send all your solutions (code, resulting images, slides) in a ZIP archive to bauckhag@bit.uni-bonn.de
- **note:** carefully debug your code! Make sure your programs do what they are supposed to do! Programming skills are essential for this course and are expected. Flawed programs, i.e. implementations of games that violate the game rules, imply failing the course.
- Remember that you have to successfully complete all three practical projects (and the tasks therein) to be eligible to the written exam at the end of the semester. Your grades (and credits) for this course will be decided based on the exam only, but –once again– you have to succeed in the projects in order to get there.
- Not handing in a solution implies failing the course.
- Your project work needs to be *satisfactory* to count as a success. Your code and results will be checked and your presentation needs to be convincing.
- If your solutions meet the above requirements and you can show that they work in practice, they are *satisfactory* solutions.
- A *good* to *very good* solution requires additional efforts especially w.r.t. to elegance and readability of your code. If your code is neither commented nor well structured, your solution is not good! A very good solution requires additional efforts towards the quality of your project presentation in the colloquium. Your presentation should be well timed, consistent, and convincing. Striving for very good solutions should always be your goal!