How to search for, find and record “related work”

Přemek Brada

2022+

Stephen Taylor

2025

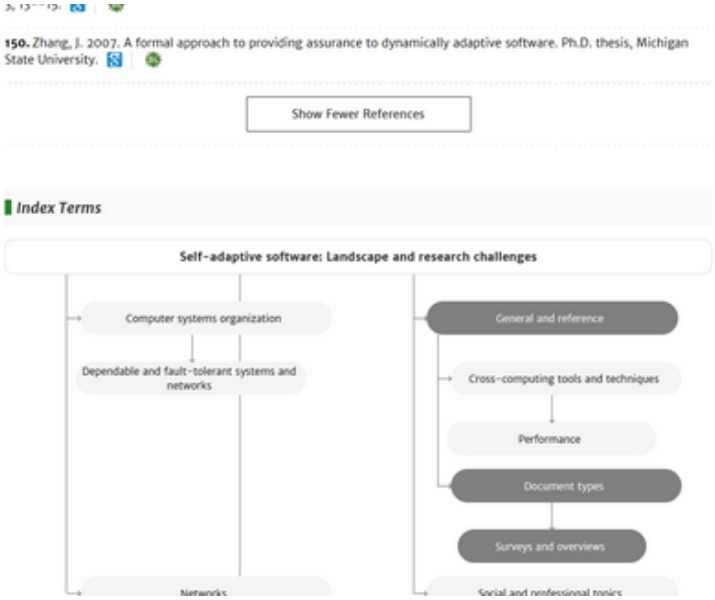
My name is Stephen Taylor. I am a researcher in the Natural Language Processing group. Before I came to Czechia in 2017, I was a professor in Massachusetts, one of the six states in the NorthEast corner of the USA which we call New England. Before I got my PhD I worked for 30 years in the computer industry, and during ten of those years my wife and I also ran a dairy farm, milking 30 cows twice a day.

As a Master’s student, you already suspect that you want to work in the field, perhaps as an academic, perhaps as a practitioner, and you are here in the university to arrive at that work with a running start.

Today I am going to talk about ‘related work’. You may have met those words already, because almost every Computer Science paper contains a section with that name. In some other fields, the corresponding section of a paper would be called a ‘literature review’. But the research to find related work is not just to fill out an expected portion of a paper.

“Related Work”?

- State of the Art
- Related Knowledge/Results
- List of references



One of the facts about our field is that it is constantly changing. The machines we work with are versatile and have important roles in almost every industry. A significant fraction of the world’s workforce is engaged in adapting them to particular enterprises or new applications.

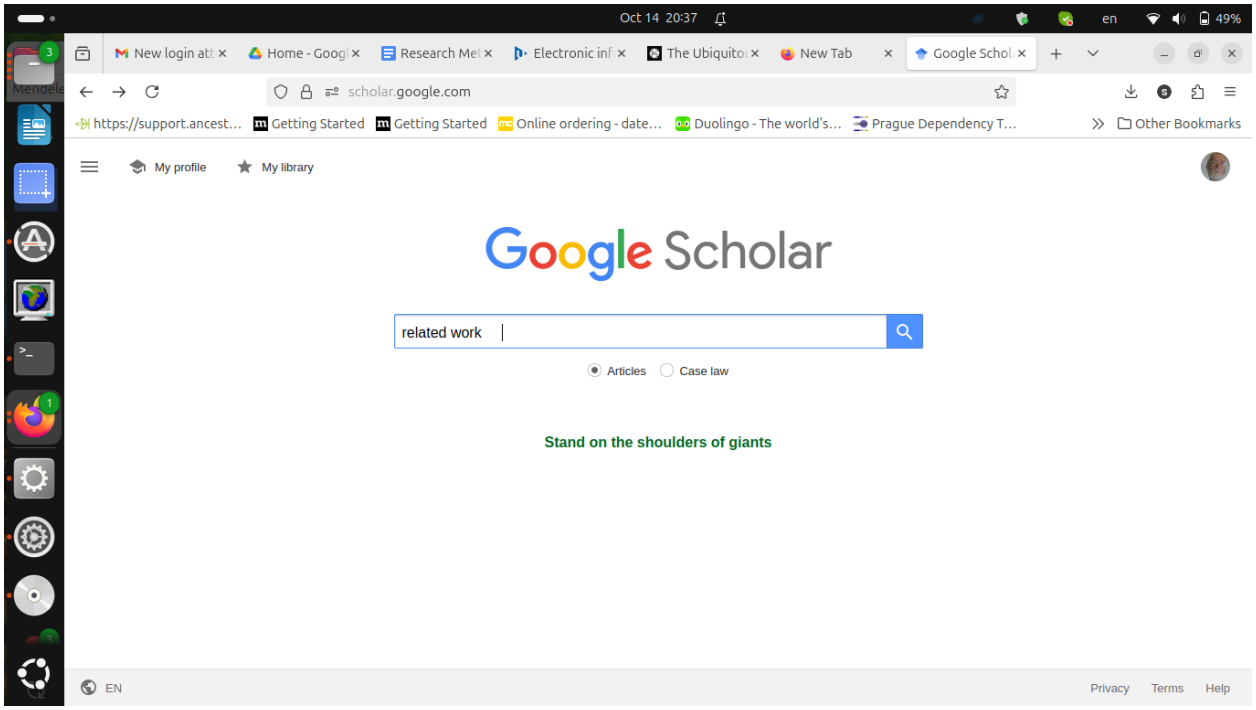
So when you start a new project, it’s good to review what has been happening in the world, to see what others have been doing in the same area. That way you can work to expand the view from their shoulders, instead of their toes. [Comment based on one by Isaac Newton: “If I have seen farther than others, it is because I stood on the shoulders of giants.”]

Searching for related work is also important at the end of a project, when it comes time to explain what you’ve done, and place it in a global context.

And it's useful to provide a backdrop against which to evaluate other people’s projects also.

As it says on the slide, the ‘state of the art’ for a particular topic is exactly what you want to discover in your search for related work. That ‘state of the art’ consists of the current best practice, but you are going to look for written material, not for actual work artifacts. In some other fields this would be a stupid approach. If you were writing about shoes, for example, it might make sense to actually look at some, and see what’s for sale. Most people won’t write a scholarly article about new uses of plastic in manufacturing shoes. They often treat this kind of fact as *Trade Secrets.*

In the business world, there are trade magazines, which give you an idea of what people are trying to sell.
In the academic world the transactional currency is articles. Articles in journals, as conference papers, as class assignments, in blogs on the web.
Most of the related work you will find out there will be in the form of articles, occasionally from books, often found on the web. That is, your search will result in a list of references.



The Web is an important tool for finding both advertising and articles, but to begin a search on a particular topic, it's helpful to have a little vocabulary, and maybe a visualized taxonomy of how the topic fits in with others. Of course you'll learn more about this as the search progresses but you aren't exactly starting from zero. You already know quite a bit.

This slide shows a search on google scholar which isn't completely stupid, but turns out not to work. Google scholar is a web server for a database of articles and authors. The slogan under the search box is based on a quote from Isaac Newton:
'If I have seen further than others, it is because I have stood on the shoulders of giants.'

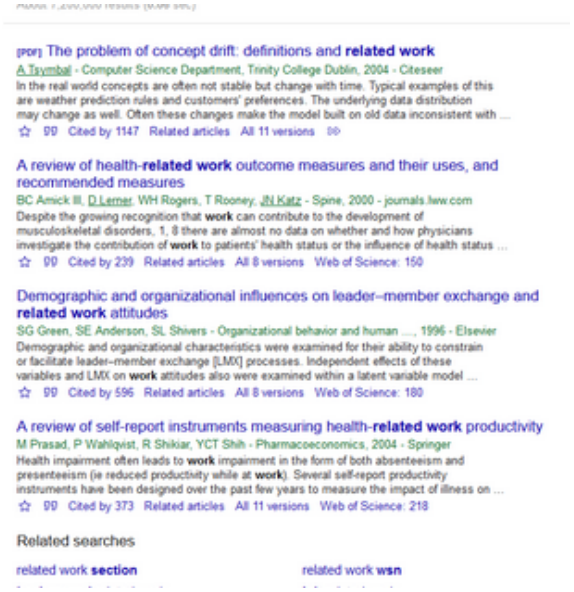
I think he meant the scholars and especially astronomers who had measured and worked out the motions of the planets, Brahe, Copernicus, Galileo, Kepler.

That's the point of the search for related work; to find the (publications) of those who have previously considered the same problem.

If you try this search yourself, you'll notice that google search tries to prompt you to figure out what the work should be related to.

Where does it come from?

- Begin at the beginning…
  - discipline / area
  - *research questions*
- Where to seek
  - *previous knowledge*
  - books and journals (uni library)
  - IEEE DL, ACM DL, Springer Link, Elsevier, …
  - Google Scholar
  - ResearchGate
  - generic search engine
- What can be found
  - Open vs Closed sources, OA
  - *seminal and survey papers*
  - grey literature
  - datasets



Sometimes you already have a particular research question you want to answer. This can happen in coursework, but it also comes up for practitioners, i.e.
'Is our server already working at capacity, or can we add more functionality without having to upgrade it?'
'Could Large Language Models improve our customer service?'

If you were writing a paper on a program you just wrote for playing TicTacToe (piškvorky) you'd probably want to compare it to other programs other people have written for the same task.

This slide is pretty dense, and I want to interpret some of the items on it for you.
The right-hand side of the slide is part of the google search output for the query we just saw. Most of these have the two words, *work* and *related* in the title. You'd probably be too disgusted with this result to make it all the way to the bottom of the list, but as we see there, google has some better suggestions for the original search.

The list on the left is more immediately relevant – it's suggestions for places to look for related work, and what kinds of items you are hoping to find. I'm going to talk about those items and places:
First the items:
Open vs. Closed usually refers to software sources. But it can also apply to data; Companies often treat both as proprietary trade secrets. You aren't likely to find those trade secrets on the web, but if you are working for a company, you may well have access to their trade secrets. You might also be able to work them out: For example, it is possible to *uncompile*, or work out a source for a binary file. Wherever you found them, if you publish trade secrets you may expect legal action.
OA, or Open Access Research, refers to software and data which researchers make available for others to build on. That's a useful and friendly thing to do; you should consider it for your own work.

Seminal and Survey papers are the gold in your search for related work. Seminal papers are the first publications of major advances. Survey papers cover a particular area, and often have bibliographies for that area which were complete at the time of publication, thus leading you to the seminal papers.

White literature includes, in addition to the gold papers, most of the rest of published output – not-so-exciting papers, books.

Grey Literature refers to items which have not gone through peer review: it includes theses; government publications; business white papers; popular literature; software and hardware manuals; the vast majority of web pages; wikipedia articles.
Obviously some of this may be very useful; it certainly tells you what is going on in the world. You are likely to use some of this information, and if you do, you should cite it.

Datasets may describe real-world data, or they may be synthesized. In my area, natural language processing, there are several large datasets openly available: various Corpora, or large bodies of text, for example the Czech National Corpus; WordNet, a semantics database, originally for English, but now covering dozens of languages; single language and bi-language dictionaries; word embeddings for many languages.

In addition there are benchmarks for particular problems; if you choose to work on one of these questions, these datasets are invaluable for comparing your work to your forerunners.

Places to look:
Previous knowledge. This is emphasized, because it is likely to be your most important asset. Luckily it grows as you work.

Books and journals, the University library. When I was in grad school, back in the paper age, my university library had hundreds of periodicals in the reference room, and back in the stacks, they had bound versions of those same periodicals. Among those periodicals were e.g. The Communications of the ACM, the Journal of the ACM, ACM Computing Surveys…

Today, your access to journals is electronic and on-line. The ZCU library website, knihovna.zcu.cz, has, in addition to the online catalog, a page called RESOURCES with links to internet hubs: the ACM digital library (which you can approach, but not necessarily access as dl.acm.org) contains links for all those journals I just mentioned, as well as many more; the IEEE digital library gives access to IEEE publications; there are several databases which contain links for hundreds of journals, probably including those I just mentioned. This is convenient if you want to work when the library is closed, and you have access to a lot more periodicals, but only if you know what you are looking for. Browsing is a lot more work than in the paper age.

The IEEE digital library and the ACM digital library are accessible from the library site. If you go to them directly, they will challenge you to tell them your institutional affiliation, and after you

succeed in spelling university of west bohemia, or only sometimes zapadočeska univerzita, they will ask you to login with your university email.

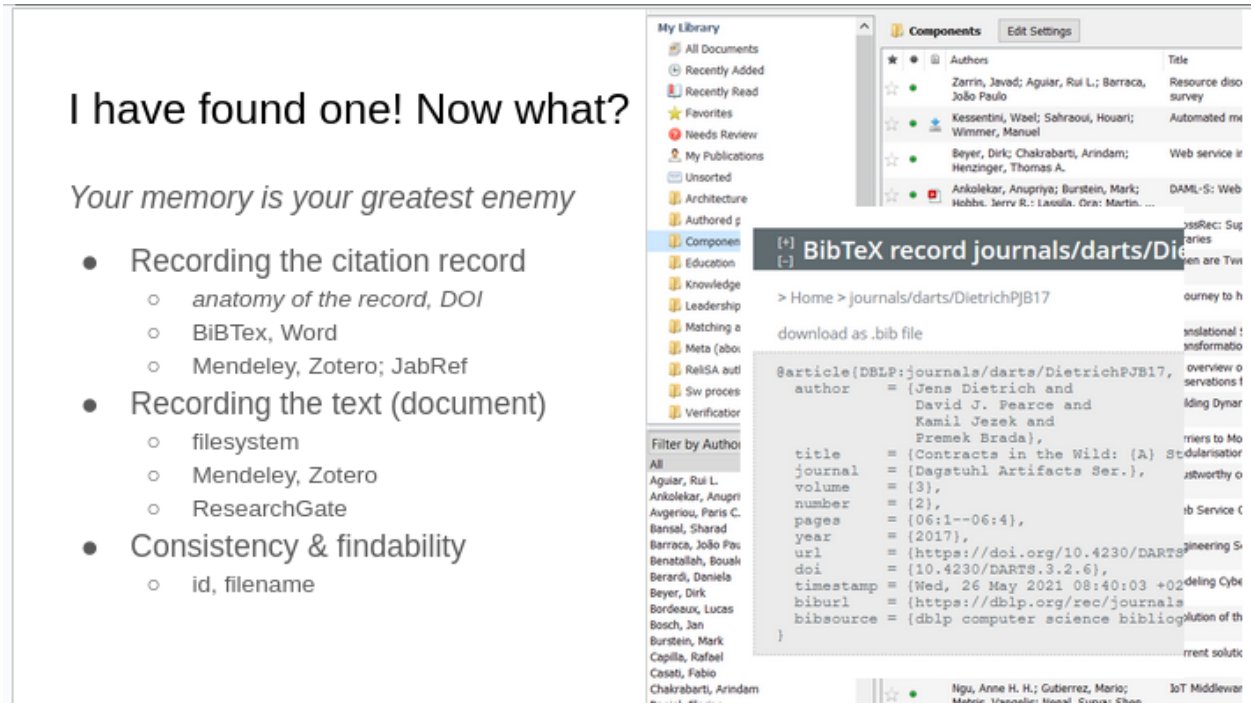
Springer link and Elsevier are set up to sell. They will quickly find articles published by Springer or Elsevier, and offer to let you download them for a price, typically 20-50 euros, but sometimes much more. Often, the authors of these have copies of these items on their own or their university websites, and you can download those for free.

scholar.google.com is an excellent tool, since it mostly knows only about papers and their contents and their authors. It is also pretty good at telling you about free downloads. But you may need to figure out key words for your particular area, even if you are willing to skim over the abstracts of not-so-relevant papers.

ResearchGate is another spot many people put free downloads of their papers. In my view, it is more author directed than google scholar, which means that it isn't quite as complete.

Regular Google Search / or Bing / or DuckDuckGo, etc. can sometimes be better than the literature-only sites, precisely because it will give you non-academic information, that Grey Literature mentioned earlier. For example, I was searching for information about bit-fields in game programming, and google search sent me to a wikipedia article on **bitboards.** That word led me to a couple of dozen articles.

Keeping organized notes



When I was in grad school and began to write my papers in LaTeX, I settled on the organizing scheme I still use: for each project, I write a BibTex bibliography file.

You see an example BibTex record on the slide.
Each item goes into a description beginning with an at-sign, followed by one of fourteen entry-type-names, and a record enclosed in curly-brackets consisting of a citation name followed by a comma, and some comma-separated fields consisting of field-name equal-sign field-value. The fields can occur in any order, but each entry-type has a list of field-names it expects, and if you omit an expected field, the BibTex software will give you an error message when you attempt to use the file to build a bibliography.

As I find interesting artifacts, (mostly conference papers, but BibTex descriptions can stretch to include almost anything, including items that probably shouldn't end up in a bibliography because readers won't be able to follow them up – for example, a photo of my father in his World War II US Navy uniform) I enter a description in the file. You can also add new fields to BibTex descriptions, and all of mine include a `notes = {}` field. I usually don't fill in the notes field until I've had a chance to read the article – but for a photo of my father, I might write the motivation right away, since I could forget the reason I mentioned it.

```
@misc{AlvaTaylor44,
   title = {Alva in his Navy Blues and Charlotte in her wedding dress},
   location = {Dining room on top of Terry's bookcase},
   year = 1944,
```

```
   notes = {\par Compare this dress uniform with one appropriate to
actual work – for example \newcite{AlvaTaylor44a}.}
}
```

If the photo actually turns out to be useful in the project or the paper, I might turn it into an illustration. Of course, to do that I'd have to scan or take a picture of it.
However, the BibTex software wouldn't help me with that; I put this item here for my own benefit. Even though BibTex is willing to put a `misc` item into a bibliography, no reader is likely to be able to make any use of this information to find it.

For this google search on writing TicTacToe programs, it looks like those Web articles are going to be a significant part of the current state of the art (which probably hasn't added too many new features in the last 75 years.) So if I choose to incorporate them into my mythical paper, I need to write a BibTex entry for each web item, and because web pages can change, the bibliography entry should mention the time I downloaded the page, as well as the URL. BibTex has a 'note' field that it will copy into the bibliography, a logical place for a tidbit like 'accessed 15.10.2024'. This field is not the same as the 'notes' field that I use as messages to myself. Notice that I write the notes field in LaTex. That's because I have a helper program that makes a LaTex file that includes all the references and notes.

For a paper, I don't usually fill the notes field in when I enter it into the file, because I often haven't carefully examined it yet – sometimes it is coming from the bibliography of another paper, and I haven't even seen it yet, but the title, or the citation, looks interesting.
As I read the paper, often much later, I try to put the high points, and also maybe things I don't like about it, into the notes.

When I download a paper, I try to include a public `URL={...}` field if there is a public link, and always a `locally={PATH}` field so that I can find the copy I downloaded. Usually it is in a papers folder near the bibliography file, but as the paper gets closer to finished, the final bibliography file (or possibly a link) will be in a directory with a .TEX file, and that may not be near my initial research.

Example BibTex Entry (partial)

```
@inproceedings{Conneau2018,
title={WORD TRANSLATION WITHOUT PARALLEL DATA},
author = { Alexis Conneau and Guillaume Lample and
Marc'Aurelio Ranzato and Ludovic Denoyer and Hervé Jégou},
year = 2018,
booktitle = "ICLR 2018",
url  = {https://arxiv.org/pdf/1710.04087.pdf},
locally= {1710.04087.pdf},
notes = {\par builds an unsupervised cross-lingual transformation using
a discriminator -- which attempts to guess whether a vector in the target
space was originally from the source space -- and a mapper, which attempts to
thwart the discriminator by adjusting the transformation;
The discriminator is a neural network with two 2048-element hidden layers.
A number of other concepts are introduced:
\begin{itemize}
\item
a eq 5: $r_T(W_{x_x})$ a mean similarity of a source embedding to its target
neighborhood and
\item $r_S$ a corresponding concept for the similarity of a target
```

This BibTex entry is for a paper which uses similarities in the structure of word embeddings in different languages to find semantic mappings from one language to another. I'm planning to test their technique on word-pairs which are translations in at least one sense, but not all, to see whether I can tease out the unmatched difference, and perhaps discover cross-lingual matches for senses missing in the other language.

Some neatniks use indenting when writing BibTex, but you can see that I don't.

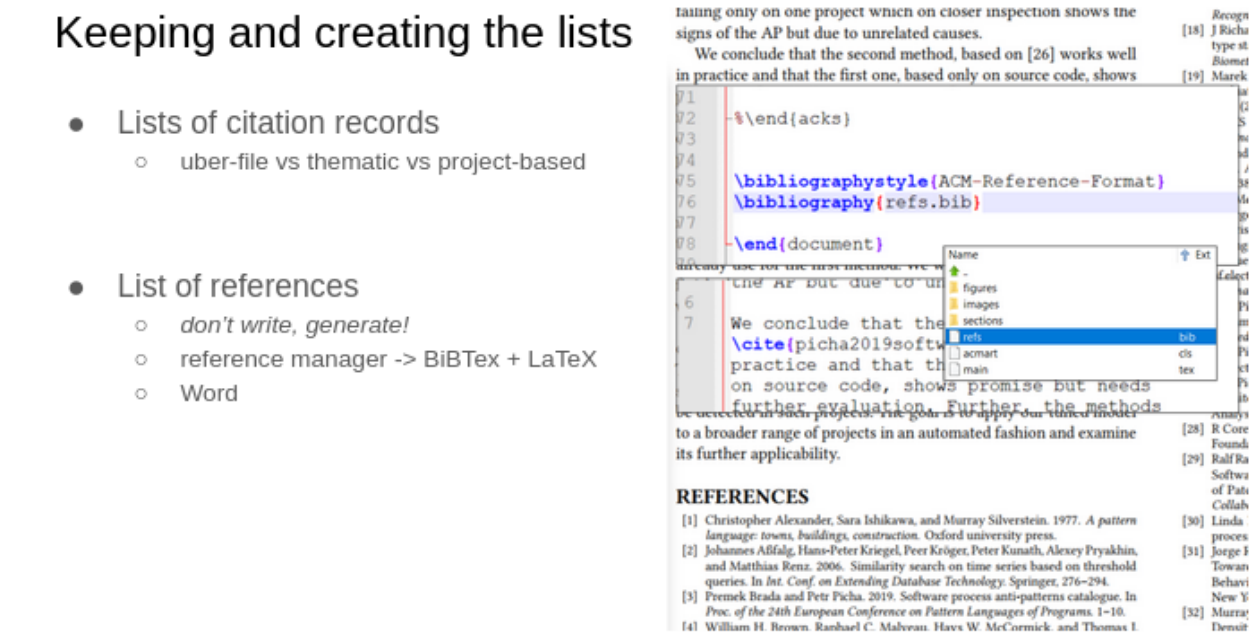
Přemek's slide mentions three more free reference organizers, Mendeley, Zotero, and Jabref. Mendeley and Zotero can keep their reference libraries in the cloud, which is handy if you are working on something both at home and at the office. Jabref seems to be a way to organize the various editing operations on BibTex files into a mouse-and-input-field app.

I spent a couple of hours playing with Mendeley. It seems to offer the possibility of gathering references for a project, then merging them into a global reference library when the project is finished. But I didn't spend enough time to get graceful with it, so you'll have to judge for yourself whether it matches your style better than text-editor + BibTex. I started out a little prejudiced against it, solely because it is said to be well-integrated with MS-Word.

My opinion about Word is (1) it isn't free; (2) Writing math in Word is painful, and the results tend to be ugly. (3) It's hard to write helper software for Word documents, partly because Microsoft seems to change the file format every other decade.

Using found bibliographies to extend your own

Before we move on, if you've found a related paper, its bibliography is going to list work that its author found relevant – and you might find it relevant to your needs as well!



This slide talks about how you might organize your reference file(s)

1) Just one file – after all, text search is fast
2) By theme, like word-embeddings; dictionary definitions; …
3) By project

I claim to organize by project, but often I append a project's file to a global file as the project finishes. I can see that there could be saved energy in Mendeley's global list with groups – but I'm not sure yet I know how to do it.

It also talks about the original purpose of BibTex – to transform a list of citations (and of course, a .bib file) into a latex file containing just the cited items, fleshed out and sorted in the desired manner. BibTex also expects a style file, which specifies which order the fields go into the bibliography entry, how names are formatted, how the bibliography is ordered, etc.

In order to support MS-Word, Mendeley provides some kind of plugin to Word to detect citations, and (it seems to me) software duplicating a fraction of the BibTex options, which produces word-appropriate output to add to your file.

You should want this functionality! I sure do! But it's only marginally relevant to the process of collecting that related work.

Long Example

A quick example

Related work for a paper about a program that plays TIC-TAC-TOE

Artificial Intelligence / Game - Playing / Algorithms

There was a chapter about this in the textbook when I took AI…

Maybe the university library has a copy…

Data structures / Bit-Boards: current state is pair of sets ({X-moved},{O-moved})

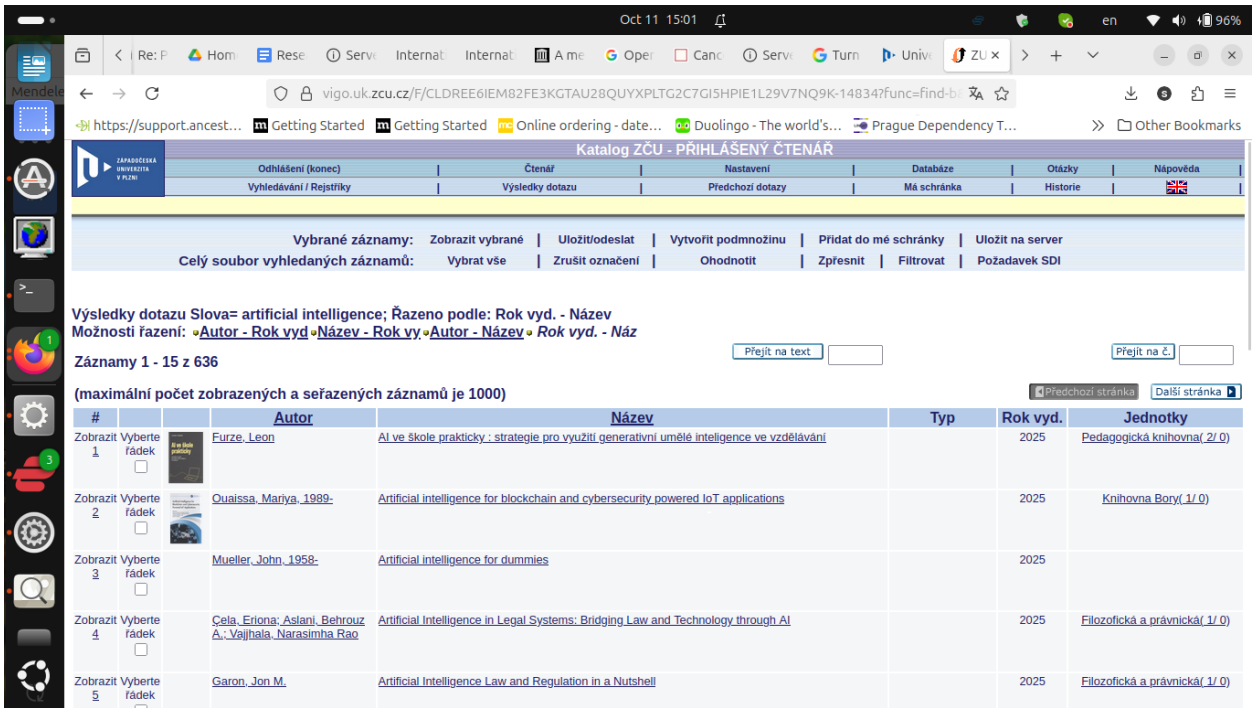
I bet most people use arrays…

I decided to wander partway through an extended example, in which I propose to write a program to play TicTacToe. This isn't part of my real research plan; for one thing I've not only already done it, but assigned it to second-semester undergrads. But this morning I woke up with an idea for a data-structure to keep the current state of the board that I didn't use before: A set of moves that X has made, and a similar set that O has made. Since there are only nine possible moves, we can store each set in an integer, which has a bit turned on for each move which has been made. If we use the low-order bits, the set which has every move in it is the number 511. But since the players alternate, the most elements that can be in the first players set is 5, and in the second players set, 4. (9 choose 5) = (9 chose 4) = 126, so that there are only 126 interesting sets for either player at the end of play, and the second players set is completely determined by the first players set, so there are only 126 final boards.
This scheme has a name; it is **bit-boards**.
The winning lines are also sets, and bit operations like X & W1 == W1 (meaning W1 is a subset of X) are cheap.
We ought to be able to pre-compute, and store in arrays, such useful facts as:

1) The number of elements in the set with this index; the place value of its least binary digit
2) What the set of possible winning completions is for the current player – that is, the set of single moves which would win the game on this turn. Some of these may be already blocked by the other player, and Wins[Me]&(511-Other) ==0 means there aren't any instant wins available.

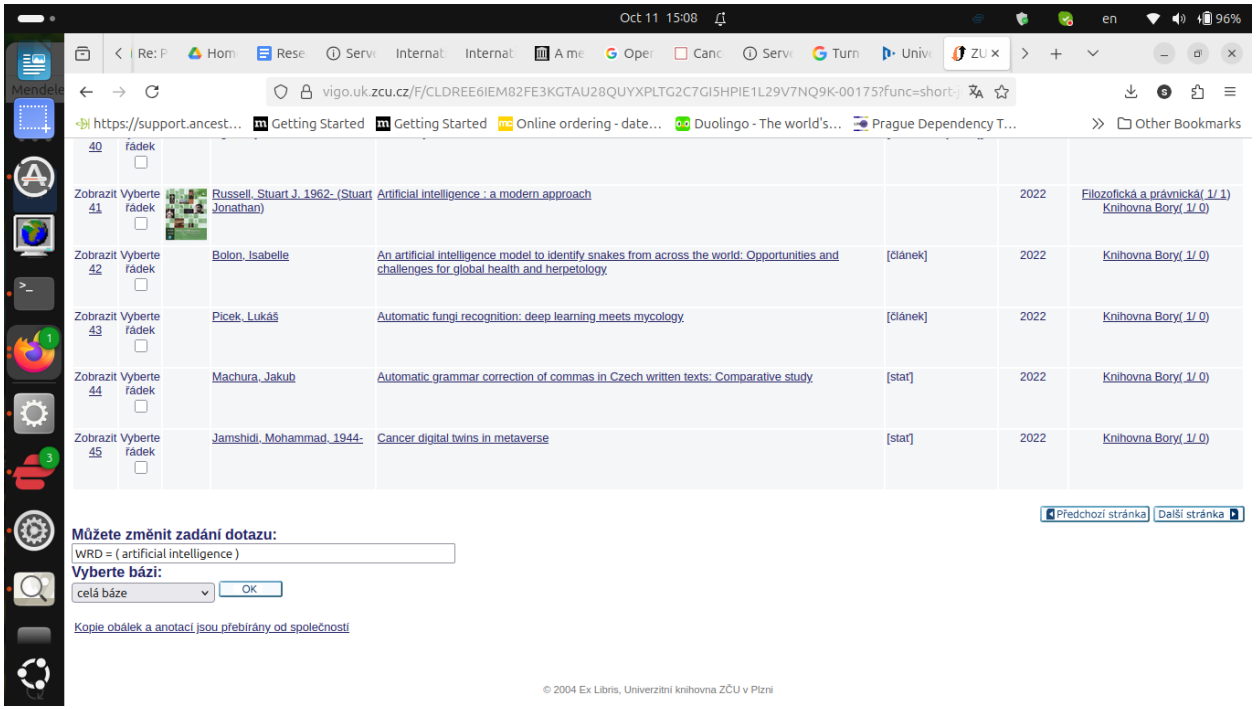
3) If I can't win, what should I block? Wins[Other]&(511-Me) != 0 means there are moves possible for the other player on his next turn which I should block.
4) Maybe possible trap moves, that is, the first move of a two-move sequence which might win.

So *maybe* an AI textbook or a big algorithms textbook. You might have those, or you could check the library catalog, and see what you can find. I tried the ZCU library, looking for books on "Artificial Intelligence", and got 656 hits. Interestingly, the fact that I was typing in English didn't seem to restrict this search much… or maybe it did.

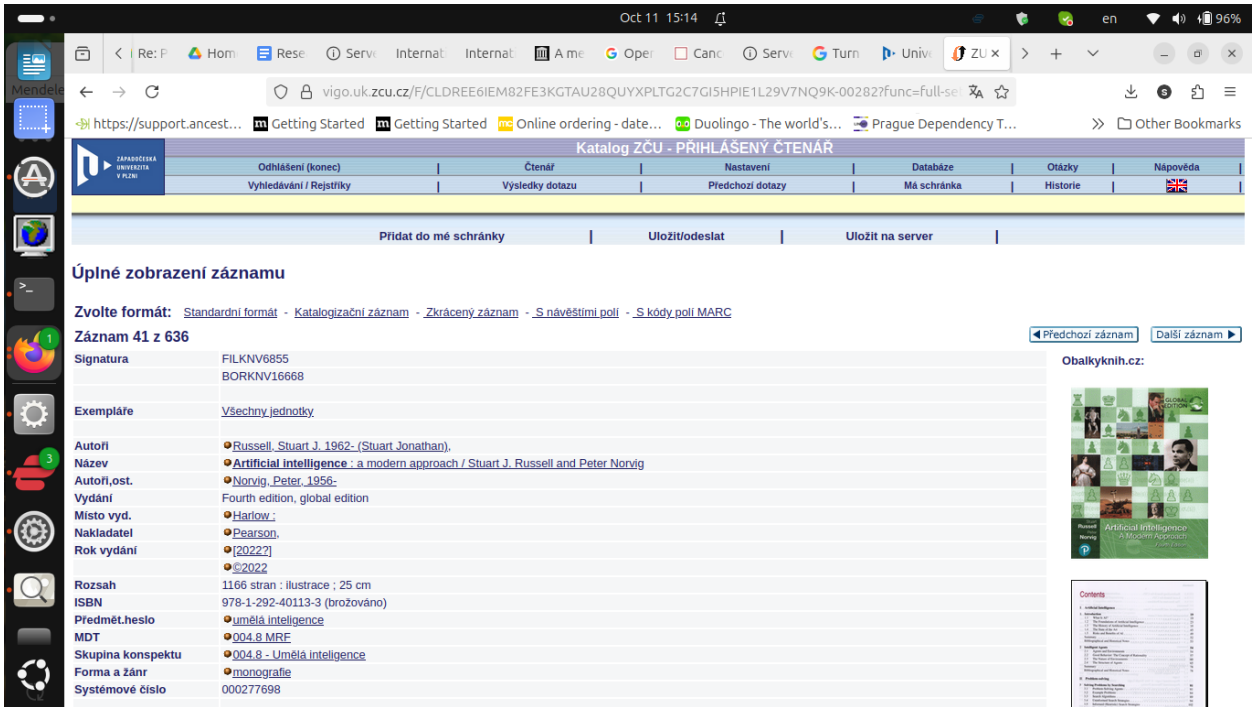


The library catalog lists the entries youngest first, so the first page of the search results lists books published in 2025. All of the 2025 books were on Large Language Models, which isn't obviously relevant. Many of the 646 are articles – one on recognizing snake species, which I thought might relate to one of my other questions … I'm pretty sure that the library has AI textbooks among its holdings, so I keep going.

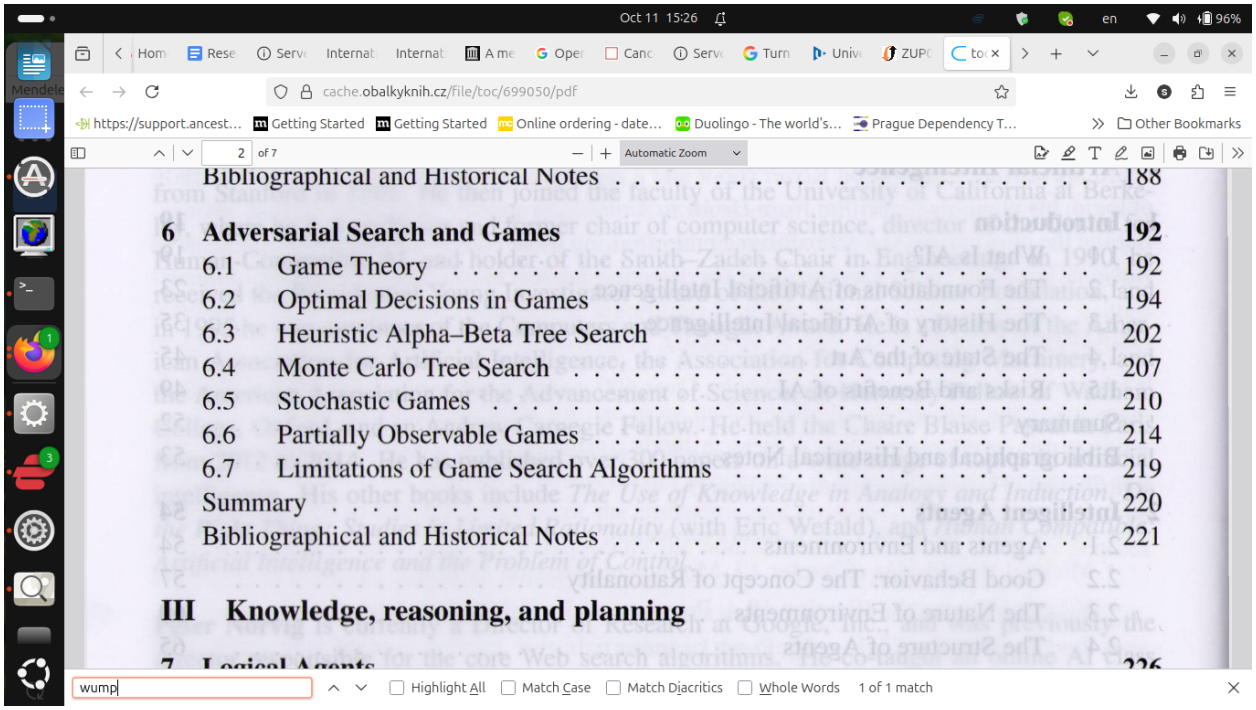
Skipping over obviously irrelevant items like "The legal status of Artificial Intelligence in the EU",



And I was right! I've used Russell and Norvig before – but it must have been an earlier edition than 2022. I'll click on the title to dig down…



And I can click on the contents to see whether my idea will pan out…



Well, this looks promising. I'd temporarily forgotten that 'game' is a technical word in AI, and applies to multi-player situations we might not call 'games' in real life, but I recognize the Alpha-Beta algorithm, and I bet that 5-pages of bibliography and notes will contain something relevant.

Too bad it's Friday night. I could either goof off for a couple of days until I'm on campus again, or try and find something on the web.

Well, wikipedia will certainly have something, especially now that I have the name of the alpha-beta algorithm.

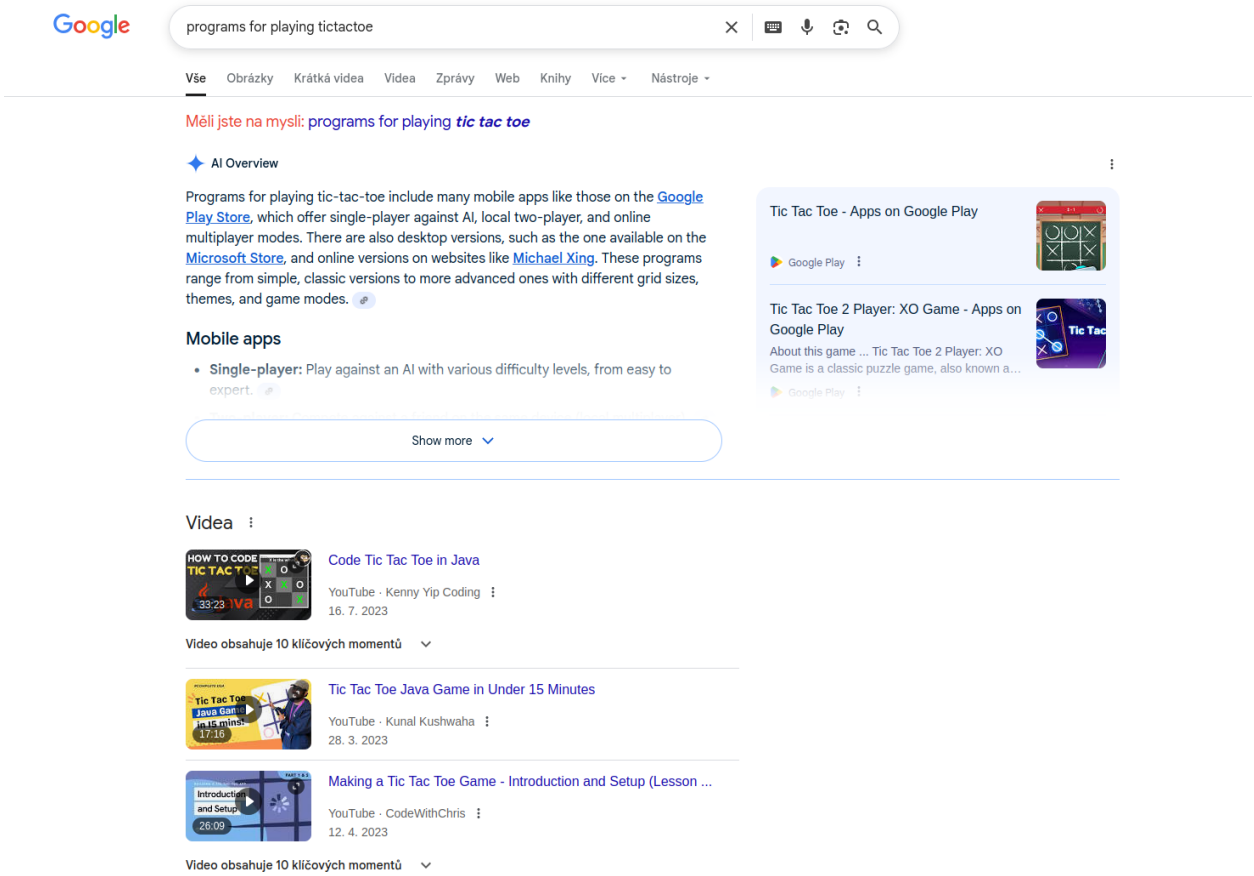
Some people say that citing wikipedia isn't good form, because although it may cite recent research, it is a *secondary source.* Wikipedia has a policy that an article should not contain original research, but should instead document results already reported elsewhere. That's why the articles always have fairly extensive bibliographies. And it's hard to cite an author on a Wikipedia article, because many people may have edited it, and most of them will have rather opaque nicknames. For example, the Alpha-Beta Pruning article has been touched by eight different people since the beginning of 2025, only one of which has a nickname that might help you find him in the offline world. On the other hand, if the wikipedia article helps me find another source, and I read it, like it, and cite it, it does seem like directing my hypothetical readers to that Wikipedia article I found useful is also a friendly thing to do.

Citing a textbook has slightly different problems. The authors are easy to find, but the Russell and Norvig textbook has gone through many printings, so it isn't obvious when they wrote a particular chapter. Chapter 6 may be describing the state of the art as of the first edition in 1994,

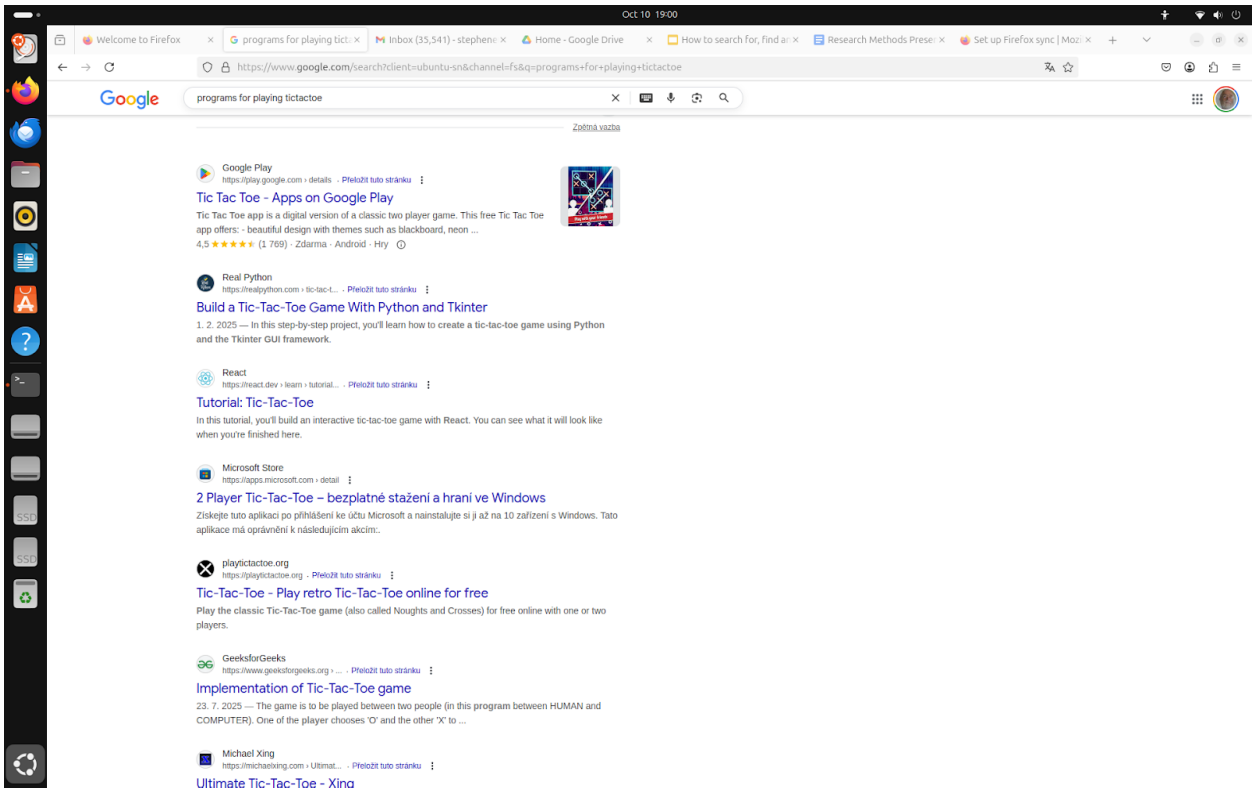
and I can't tell unless I check that bibliography. But citing a textbook is pretty common. Encyclopedias and textbooks aren't primary sources. They are both surveys or distillations of knowledge, or *secondary sources.*

So I do a google search for Alpha Beta algorithm, and the top result is a wikipedia article on 'alpha-beta pruning'. As I thought, Alpha-Beta is a scheme for choosing moves in a two-player game, or more precisely ignoring some of the choices on the tree of possible moves, allowing a game-playing program to decide on the next move more quickly. But the problem is, it is impossible to win TicTacToe unless your opponent makes a mistake. So if we assume the opponent will not make a mistake, a deep search of the possible moves isn't much help, and not only alpha-beta pruning, but all the tree search algorithms are a bit of a waste of time. (Of course, the search could help the computer avoid a mistake, but if we grab any available 1-move win, and if no win is available, try to block opponent wins, we're already playing at the seven-year-old level, and that might be good enough.

So we could go to google search to see about comparing data structures with other TicTacToe programs, and I might as well start with a fairly specific query, 'programs for playing tictactoe'



Here’s the first page, dominated by an ‘AI summary’ which might be intended to discourage you from reading the second,



The second page includes more of the same.

BibTex file, with notes ….

```
@misc{Zaczyński,
    author = {Bartosz Zaczyński}
    title = {Build a Tic-Tac-Toe Game Engine With an AI Player in Python},
    publisher = {Real Python},
    URL = {https://realpython.com/tic-tac-toe-ai-python},
    accessed = {13 October 2024 15:10},
    year = {?current -- text says python 14 is coming.}
    notes = {\par
        This is a project perhaps designed for teaching usage of python immutable classes.
        The code is majorly wordy and keyword-filled.
        The state datastructure is basically a class which holds an immutable 9 character string.
        The 'AI gameplay' module performs a minmax tree search, going all the way to the bottom of the tree
        on each move.  I think that each possible move will send a new state object to the next level, so
        there is a lot of allocation while doing so.
    }
}
```

```
@misc{Kenyon,
    publisher = {thesharperdev.com},
    author = {Morgan Kenyon},
    URL = {https://thesharperdev.com/coding-the-perfect-tic-tac-toe-bot/},
    year = {?},
    title = {Coding a perfect Tic-Tac-Toe Bot},
    accessed = {13 October 2025 15:58}
    notes = {\par
        The data structure is called a grid; it is a 2-d array of (one-character) strings.
        The game makes each player a class with methods, so two programs can play against one another.
        Several players are presented:
            The random player makes legal moves, but at random.
            The one-layer player grabs one-move wins.
            The two-layer player also blocks next-move wins by the opponent.
            The invincible player uses minimax tree search, said to take about 30 seconds per move.
        Ranking of players about as shown.
        Two-layer player occasionally loses to random player, indicating that strategy not sufficient.
        Invincible player was only tested against random player, so we don't know whether it would always beat
        two-layer.  But since two-layer occasionally loses to random, probably it would always lose to Invincible.
    }
}
```

```
@misc{Heaton3ab,
	author = {Robert Heaton},
	title = {Programming Projects for Advanced Beginners #3a,#3b: Tic-Tac-Toe AI},
	year = 2018,
	URL = {https://robertheaton.com/2018/10/09/programming-projects-for-advanced-beginners-3-a/},
	publisher = {robertheaton.com},
	accessed = {13 October 2025 17:36},
	google_search = {programs for playing Tic-Tac-Toe}
	notes {\par
		Data structure is a 2-d array of strings or None.   Each move creates a new array, so the states are
		immutable, probably interesting for minimax search.
		Talks about the same mechanized players as \cite{Kenyon}, but also adds a series of speed improvements
		as possible extensions, including caching, and using the same cache key for rotated and reflected variant
		states.
	}
}
```

I looked over several of these ‘tutorials’ – many are for two human players, and the program is used as a combination of ‘substitute for pen and paper’ and ‘referee’.

However, I wrote up several which actually have one human and one AI player. The ones I wrote up all seem to use 2-dimensional arrays of character strings to hold the current board, This means that each node analyzed in the minimax tree will have four list objects allocated. In competition, the pairs of integers used in my proposed technique (which I haven’t implemented to measure) need one integer object allocated per recursion, which should be a big speedup. Possibly one could avoid allocating the integer object by passing a mutable bitstring or ‘buffer’ object. Switching to C is probably a more sensible plan, but that would mean I couldn’t honestly compare with the speeds of the python programs..

Several of the tutorials test the grab the quick win / block the opponents quick win strategy I thought might be enough; they suggest it is not.

So far my related work is all for python; but I should change that.

The main disappointment with this example is that these web pages, which *could* have lots of interlocking links, don’t reference related work.