

# Project Description: CSCI 350 Spring 2024

**Albert is right — it is all about the generator...**

You will find, posted on the class website, very good advice from the last AI class on how to address the project. **Take it seriously.**

## 1. Goal

Build an expert Mastermind player in Python to compete in our 100-guess, time-limited tournaments. **Don't program or start surfing the Web yet. Keep reading.**

In the basic 4-6 version of this game, your player tries to guess a **concealed code**, a 6-peg sequence made from four colors (denoted here by the letters {A, B, C, D}). There are of course  $4^6$  possible 4-6 codes, including AAAABA, ABAAAB, and DBCAAA.

Your program will not just be playing the 4-6 version, however. The game can (and will) be made arbitrarily more difficult by increasing the number of pegs and the number of colors. Your player must compete for any (positive integer) number of pegs  $p$  and any (positive integer) number of colors  $c \leq 26$ , represented as distinct consecutive letters of the alphabet in lexical order, beginning with A. (For example, if  $c = 4$  the colors will be A, B, C, and D.) If your opponent were irrational (i.e., random) this would be hopeless; a 10-12 game, for example, would have  $10^{12}$  possible codes but you will still get only 100 guesses for any values of  $p$  and  $c$ . Your team will write a Mastermind player that can *play any size game well*.

To test and evaluate your player, we have built multiple **tournament engines**. Every time your player makes a guess, our tournament engine returns three numbers: how many exact pegs (correct color in the correct position), how many “almost” pegs (correct color in the *wrong* position), and what number guess you just made. For example, if the reply to your guess BBACDC is 2 1 39, it means that on your 39<sup>th</sup> guess two of those pegs have the correct color in the correct position and one other peg is the correct color in the wrong position. (Ah, but which are they?)

## 2. Your opponents

Because this problem is NP-complete, we built you some rational agents to play against. These are the Secret-Code Selection Algorithms (henceforward, **SCSAs**). Your program will know the name of the SCSA you are playing against. Your algorithm should be parameterized to work against each SCSA *by name*. (As a result, any learning you do should be offline, *before* the tournaments. You won't have time to learn much during a tournament.)

An SCSA is a class with a function that creates and returns hidden codes when given the number of pegs, list of colors, and number of codes to generate. For example, the following asks for one 7-peg code. It uses the letters A,B,C,D to represent colors and could return ACACACA:

```
scsa = TwoColorAlternating()
code = scsa.generate_codes(length = 7, colors = ["A","B","C","D"], num_codes = 1)
```

You will be playing against **15 different SCSAs**. Eight are fully observable, 5 are partially observable, and two are completely hidden from you until we run our tournaments and your player encounters their codes.

- **Eight SCSAs have visible code generators:**

InsertColors	TwoColor	ABColor	TwoColorAlternating
FirstLast	OnlyOnce	UsuallyFewer	PreferFewer

You can generate a set of codes from any one SCSA and write them to a file so that your player can practice on them. The file name is built from the parameters you pass. The following example generates 25 7-peg codes on 5 colors with `TwoColorAlternating` and writes them, one code per line, to a file. Note that the codes are not necessarily distinct and would be on separate lines in a file named `TwoColorAlternating_7_5.txt`

```
scca = TwoColorAlternating()
scca.generate_and_write_to_file(length = 7, colors = ["A","B","C","D","E"], num_codes = 25)
```

CACACAC	DADADAD	BDBDBDB	ECECECE	DEDEDED	CACACAC	AEAEAEA
ABABABA	EAEAEAE	BDBDBDB	BCBCBCB	BABABAB	BDBDBDB	EBEBEBE
DCDCDCD	DCDCDCD	EAEAEAE	ABABABA	ECECECE	CDCDCDC	BCBCBCB
AEAEAEA	CDCDCDC	AEAEAEA	CECECEC			

- **Five SCSAs are partially observable:**

mystery1 mystery2 mystery3 mystery4 mystery5

You cannot see their code generators. Instead, on Blackboard, there is a file of 200 sample codes for 7 pegs and 5 colors generated by each of these five SCSAs. You have our assurance that they are not purely random. You can practice on those codes too, or try to figure out what makes them tick, and use it to inform your program.

- **Two SCSAs are entirely hidden:** mystery6 and mystery7

They are not purely random but are entirely unobservable; you won't be able to practice against either of them before the final tournament.

Your team will write a Mastermind player that can *play well against any of these 15 secret-code generators*. Where's the competition here? It's who can get the highest score in the tournaments. **Read on.**

### 3. Rounds and tournaments

A *round* in a tournament pits your program against a code from an SCSA for some fixed number of pegs and colors. In each round, our system chooses a secret code and your program tries to guess that code with **no more than 100 guesses and within 5 seconds**. If you make an illegal guess (wrong number of pegs or illegal colors) the round ends. At the end of each round, we record whether or not you won (guessed the code) and, if so, how many guesses it took you to do so. (Fewer is better.) We also record whether or not you made an illegal guess.

A *tournament* consists of a set of 100 rounds against codes from one SCSA for the same number of pegs  $p$ , colors  $c$ . Every team will face the same sets of codes. We will do that with the same codes for every SCSA and for every team. There is a **5-minute total tournament time limit** in any given tournament for any player you write. **If your program somehow breaks execution during a tournament we run, we reject your program**, so you should be sure to always make a legal guess, even if it is a stupid one. No tournaments will be run against the random player; it is there only for your convenience. Of course, in the final competition there will be a fresh set of codes from all these SCSAs.

If your player times out on any 10 rounds during a 100-round tournament, your player will be disqualified from competition, but only against *that* SCSA in harder tournaments. (Don't worry...there may be hundreds of tournaments.)

In a tournament, your team **scores**  $pc(5g^{-0.5} - 2i)$  points for each win, where  $p$  is the number of pegs,  $c$  the number of colors,  $g$  is the number of guesses it took to win, and  $i$  is the number of rounds ended by an illegal guess. The program with the highest score in a tournament wins that tournament. We will initially probe your program's performance with a variety of tournaments where  $4 \leq p \leq 10$  and  $6 \leq c \leq 10$ , but expect that we will ramp up dramatically (e.g., to 10-12, 12-14, 15-20, 20-25, and probably much higher) to determine the champion.

At the end of a tournament, our output will report your tournament score and how many wins, draws, and rounds with an illegal move you had. For example, this output

```
Player: Kangaroo
Game: 7 Pegs 5 Colors
Rounds: 15 OUT OF 100
Results: {'win': 2 'loss': 12 'failure': 1}
Score: 3.14
```

indicates that in a 100-round tournament the team named Kangaroo won 2 rounds (score reduced for how many guesses it took) and lost 12 before it made an illegal guess and was rejected by our tournament engine after only 15 rounds.

## 4. Baseline strategies

The Web is filled with ideas, but then again, so are your own heads. **We recommend your heads.** You can read about Mastermind on Wikipedia here: [https://en.wikipedia.org/wiki/Mastermind\\_%28board\\_game%29](https://en.wikipedia.org/wiki/Mastermind_%28board_game%29). Feel free to read and look about on the Web and draw inspiration from it. You won't find anything out there, however, that will do well for the tournaments defined here. Promise.

Here are 4 **baseline strategies** that may not scale either, but they are good baselines against which you can compare the performance of your far more sophisticated strategy.

**B1:** Exhaustively enumerate all possibilities. Guess each possibility in lexicographic order one at a time, and pay no attention to the system's responses. For example, if pegs  $p = 4$  and colors  $c = 3$ , guess AAAA, AAAB, AAAC, AABA, AABB, AABC and so on. This method will take at most  $c^p$  guesses.

**B2:** Exhaustively enumerate all possibilities. Guess each possibility in lexicographic order *unless* it was ruled out by some previous response. For example, for  $p = 4$ , if guess AAAB got 0 1 1 in response, you would never again on that round make any guess that began with AAA or ended in B.

**B3:** Make your first  $c - 1$  guesses monochromatic: "all A's," "all B's," ... for all but one of the  $c$  colors. That will tell you how many pegs of each color are in the answer. (You don't need to actually guess the last color; you can compute how many of those there are from the other answers.) Then you generate and test only answers consistent with that known color distribution.

**B4:** The article by Rao (posted on Blackboard) has a heuristic algorithm that is probably not optimal.

## 5. How to approach this project: think first!

Since this is AI, you know that **knowledge is power**. Therefore, you should use the name of the SCSA to help you guess. If you want to win this competition, you should also think seriously about how to address the mystery SCSAs. **Don't program or start surfing the Web yet; keep reading.** For any 4-peg, 6-color code, there is a 5-guess solution by Donald Knuth. The 5-guess approach exhaustively enumerates all possible codes, and eliminates those inconsistent with the responses to the guesses as they are received. It then chooses a guess with the highest score, the one that maximizes the possibilities that could be eliminated by any of the possible

responses. Of course, **this strategy does not scale**. Like many fun AI problems, Mastermind is NP-complete <https://arxiv.org/abs/cs/0512049>.

You may use standard Python libraries. Beyond that, you must *cite your sources properly*. **See the handout on how to avoid plagiarism. You may not use programs specifically intended for Mastermind from any repository nor may you use any genetic algorithm.**

**Teams that organize to share work and ideas and meet regularly do better.** This is a challenge for your mind and for your ability to work together as a team. Moreover, in the final submission everyone will have to justify and explain exactly what they contributed to the final product in a **brief, all-by-yourself oral exam and in a written statement**. Your team may find a private repository on GitHub helpful.

**Your team faces 3 challenges:**

**#1: Implement a general-purpose player.** Your program should be able to play for any number of pegs and colors. It should also make reasonable guesses based on the responses it has received.

**#2: Implement a scalable player.** As the number of pegs and colors increases, any given algorithm will take longer to make each guess, and more guesses will be required. Scalability is measured in time and in number of nodes expanded. *You do not want to make the same guess more than once*, so you probably want to keep track of (at least some of) your guesses and work through them methodically. (You could also generate one guess at a time, but that might be hard to do methodically.) This is why the third number returned on a guess is the guess number. Feel free to define variables and/or data structures to help you remember what has happened in the current round, and don't forget to re-initialize them every time a round begins.

**#3: Learn the SCSA's strategy for choosing the secret code.** No SCSA is purely random. Your program should exploit the SCSA's code-generating strategy to make better guesses. The SCSA may only use certain colors, or always choose codes with exactly three colors, or always choose codes that alternate colors, or never use the same color more than once, or always put the same color in the first and last places, or always place colors in a certain order (e.g., A is always before B), or prefer codes with fewer colors but occasionally use more colors to mislead you, or have a probabilistic preference for fewer colors (e.g., with probability 0.5, use exactly 1 color; with probability 0.25, use exactly 2 colors, ...). SCSA strategies may include, *but are not limited to*, those described above. *Think carefully* about your learning feature space and algorithm.

Your team should design a guessing strategy that you expect to do well on challenge #1. After that, most teams will choose to focus more of their energy on either challenge #2 or #3. You should have *some* solution that is plausibly scalable (works for any size problem, at least in theory), and *some* learning approach (even if it is quite simple and limited). Teams of 3 or more people will be expected to have good designs for all three categories. *Your project write-up (discussed below) must address all three challenges and how your design tried to meet them.*

## 6. Programs: our engines and your Players

There are two versions of the tournament engine, the one we have provided on Blackboard and the one that runs our tournaments, hidden forever. The program you submit for grading must run without error on our hidden engine. To grade your project, we will load our hidden version first, and then load your submitted program. You can find a persistent assignment, Player Test, on Gradescope, where you can check if your player runs with our hidden version on various SCSAs, number of colors, and number of pegs. To test with respect to specific deadline requirements, there will also be one Gradescope version for each project deadline. We encourage you to test

earlier than each deadline so you can continue to expand your player's prowess. Read the engine we have posted for you carefully before you plunge into this project, especially the comments. The engine we provide in Mastermind.zip is a zip file; it contains files that represent the game itself, the eight observable SCSAs, and several Players. The commented-out lines in mastermind.py are print statements that will prove helpful in debugging. Uncomment and use them to gain insight and to help debug your Players.

Your task is to write a class that inherits from the Player class and implements the function `make_guess` that makes a guess at a secret code for Mastermind, one that plays well in our tournaments. The name of this class should be the name of your team (e.g., `Kangaroo`). As an example, we have included in Mastermind.zip a genuinely dumb Player, `RandomFolks`. Your baseline players should be named `Baseline#`, (e.g., `Baseline1`) where `#` is in `{1, 2, 3, 4}` in accordance with Section 4 above. **The constructor for your class should not require any arguments.**

During a tournament, the `make_guess` function takes as arguments the number of pegs, the colors that can be used in the code, the SCSA used to generate the code, and the game response to your previous guess (number of pegs that are correct and number of pegs that are the right color, but in the wrong location). You are free to write additional functions and use attributes for your class, but we will only call your `make_guess` function as specified during a tournament of Mastermind. You can modify other parts of our posted engine as you wish for testing, but **we will only use the class in your submitted file during our tournaments.** Your submitted .py file should only include your class and necessary helper functions that your class uses; it can import from any of the provided files.

We strongly recommend that you keep track of time in your Player so that you are not disqualified when you exceed the time limit for a round or for a tournament.

Test your players with `main.py`. It can be run with the following:

```
python3 main.py <board length> <num colors> <player name> <scsa name> <num rounds>
```

For example, to test `Kangaroo` in 100 rounds of a 4-5 game, execute:

```
python3 main.py 4 5 Kangaroo InsertColors 100
```

Under our score function (defined in section 3 above), this could result in the output

```
Player: Kangaroo
Game: 4 Pegs 5 Colors
Rounds: 100 out of 100
Results: {'win': 10, 'loss': 90, 'failure': 0}
Score: 50
```

Our tournament engine will run your players similarly.

There is a thread on Slack to ask and answer questions strictly related to Python. If you believe you have found a bug in our engines or have questions about it, send email to [csci350uta@gmail.com](mailto:csci350uta@gmail.com). If you have questions about the project itself, ask Professor Epstein.

## 7. Project requirements and grading

You will be primarily graded on the *thoughtfulness and clarity of your design and presentation*, and not primarily on your algorithm's performance. This gives you the freedom to try a risky approach that is interesting from a design perspective but might not work very well. An approach that does not work very well, and is also naive, trivial, or not well-motivated will receive a correspondingly trivial grade. The most important part of this project

is to *produce a working program with a strong justification and a clear design beyond the baseline strategies in Section 4* for your player. Winning is nice (and fun) but it won't earn you many points.

The following could add to more than 100 points and gives you a chance to get really involved.

**Team formation (2 points):** Identify your team of four students. Come up with a team name and enter it for every member on the team meetup sheet. Your name should be family-friendly and not too long, since you will be typing it a lot. Specify a contact for your team, one person who will be responsible for all submissions and whom we will contact if issues arise. Enter your contact for every member on the team meetup sheet.

**Baseline player implemented in Python (5 - 20 points)** 5 points for each of the 4 baseline strategies listed in Section 4. *You are only required to implement one.* Any baseline implemented for credit must be named **TeamName\_B#** where # is the number of that baseline strategy described in Section 4.

**Your team's tournament player implemented in Python (35 points):**

- 15 points for correctness (whether the implementation matches the solution described in your paper)
- 15 points for design (generality, clarity, and elegance)
- 5 points for code readability (indentation, comments, modularity)

**Project report (23 points):** Each team must submit *one copy* of a clearly-organized project report that describes your approach, your experience in designing and implementing the approach, and the performance of your system.

This must be a *minimum of 5 pages* including:

- 4 points for a **survey** of any background reading you did on the game and strategies, *with citations*.
- 10 points for a **discussion** of how you constructed guesses and how you learned for biased SCSAs. *Provide explicit citations* for any ideas you drew upon or borrowed directly from the literature.
- 9 points for some **theoretical analysis** (mathematically formal would be nice, but is not required) of the computational complexity of your algorithms, and the number of expected guesses (which could be based on the degree to which each guess is expected to reduce the size of the remaining solution space) in terms of the size of the problem.

**Your experimental evaluation (25 points)** of your program with respect to the three challenges in Section 5.

Report performance results for your program and, separately, for at least one of the 4 baseline strategies against **all** of the following:

- The random SCSA InsertColors and at least 5 of the known biased SCSAs in tournaments of 100 rounds for the 8-10 problem. **You may find this is quite slow.** If you cannot get results in a reasonable time, you may reduce this to 7-9 or even 6-8, but that means you have not thought your guessing strategy through very well and are unlikely to win.
- The scalability challenge in a series of tournaments of increasing difficulty (*pegs × colors*). You choose the problem sizes. *You will understand scaling better if you start with small values for pegs and colors and work your way up.* Do this for more than one SCSA.
- The learning challenge, using data from the SCSAs provided, both the known and the hidden ones.

**Present all results clearly** using tables and/or charts, scatterplots, and/or statistics (e.g., means, standard deviations, confidence intervals, violin plots). In each case *provide information on both CPU time and guesses*.

(Hint: Use a Python timing function.)

**Class tournament (10 points)**

- **3 points for** a program that runs successfully (i.e., no illegal guesses) during the tournament
- **7 points for** how well it performs in the tournament



## 8. Deadlines

**These deadlines are hard. No exceptions and no extensions.** Submit each project step to its respective Gradescope assignment by 11pm on the date specified on the class website. **Only one person per team should submit the files.** Be sure to mark your teammates on Gradescope when you submit. Your players will be partially autograded and partially manually graded.

For each of the following there should be at least 2 parts: **one .py file** that will run in our environment and **one .txt file** that contains the indicated output.

**All .py files should stand alone.** In other words, do not return to us any revised versions of our functions, just the ones you wrote for your **Player** and for the functions it calls.

**DEADLINE 1:** Form your team of 4 people, and list your team's name and team's contact on the meetup sheet. Every person must enter those names.

**DEADLINE 2:** Feel free to submit early for confirmation that you are on the right track.

- Player **teamName\_B#.py** where # is 1,2,3, or 4 for *exactly one* correctly-named, functioning **baseline player** that can play a **4-6** tournament in our environment without illegal guesses.
- Output (**teamName\_B#.txt**) from a **4-6** tournament where your baseline played 100 rounds *against 2 different non-trivial SCSAs* without illegal guesses. (Which ones are trivial? InsertColors, ABCColor, and OnlyOnce. What's so easy about InsertColors? It is totally random, so easy to compete against but hard to win against.)

**DEADLINE 3:** Feel free to submit early for confirmation that you continue on the right track. Upload one zipped file **teamName\_3.zip** to Gradescope that contains all of the following:

- Player **teamName\_d3.py** for your current **tournament** player that runs in our environment and makes no illegal guesses. (If you are wondering, that *d* is for deadline.)
- Player **teamName\_B#.py** where # is 1, 2, 3, or 4 for your team's baseline player that runs in our environment and makes no illegal guesses.
- Output (**teamName\_d3.txt**) from **5-7** tournaments where your current tournament player played 100 rounds in a 5-7 tournament *against each of the 13 SCSAs* and made no illegal guesses.
- Output (**teamName\_B#.txt**) from **5-7** tournaments where your baseline player played 100 rounds in a **5-7** tournament *against each of the 13 SCSAs* and made no illegal guesses.

If you are submitting **extra baseline players for credit**, upload a separate **teamName\_B#.py** and its separate **teamName\_B#.txt** with your other players to Gradescope. **No new baseline players will receive credit after this date.**

**DEADLINE 4:** Upload one zipped file **teamName\_4.zip** that contains all the following:

- Your near-final tournament player (**teamName\_d4.py**) that runs in our environment.
- Output (**teamName\_d4.txt**) from an **8-10** tournament when your near-final tournament player played 100 rounds well *against each of 2 non-trivial SCSAs*.

**DEADLINE 5:** Upload the following to Gradescope to their respective assignments:

- Your final tournament player (**teamName.py**) that runs in our environment.
- A **.pdf** file of your report with a completed version of the **coversheet** (found on Blackboard), **signed by every team member**.