Guidelines for Grades on Strategy Design Assignments

All grades are inherently a bit subjective, even in proof-based courses with a clear "correct" answer. Still, grades on strategy design assignments are inherently a little bit more subjective than the rest of the course. Below is a brief explanation of how your grades will be computed.

- **Every strategy design assignment will be worth 50 points.**
- **20 points can be obtained by "correctly" answering two short prompts given in the assignment**. "Correctly" is in quotes because the questions are not mathematically formal, and you are not expected to write a proof that your answer is "correct." You are expected to rigorously justify your answer using ideas from the course (and it may or may not make sense to include a proof of a related statement). There may be multiple "correct" answers, but there will always be at least one that can be deduced using ideas from the course. These answers will be graded as objectively as any other problem in the course.
- **15 points will depend on the quality of your solution**. There are two ways to earn these:
  - Performance of your solution. We will run everyone's submitted code, along with a few submissions by the course staff, ranging from trivial to more interesting. You'll receive up to 15 points based on how many course-staff submissions you outperform. Observe that these points are not zero-sum, as you must only outperform the course staff. **Note:** we do this run after the submission deadline, and describe how code scores were computed in the staff solutions.[1]
  - Quality of writeup and justification. This evaluation is inherently more subjective than the rest of the course and will be graded based on "how well you convinced the grader that your solution is a good approach." You should focus on *why* your strategy is good, and not *what* your strategy does (although some of the what is necessary to get the why).
  - Your score will be the maximum of these two.
- **15 points will evaluate your ability to think critically about how the particular model affected your solution**. This writeup is evaluated similarly to part c.
- There are 5 strategy designs throughout the semester. Your lowest of the 5 grades will be dropped. You may drop your first grade because you found the instructions unclear at first, the last assignment because you're happy with your first 4 grades, or any assignment during an unusually busy period. You do not need to declare which grade you would like dropped, your lowest grade will be automatically dropped (including empty submissions).

Collaboration: The emphasis of these projects are on *design* of strategies, not implementation. As such, you may use snippets of code provided by staff, or that you find online (provided that those snippets were not written with the intent of solving this strategy design exercise). You may not use snippets of code written by other students in this class or any previous iterations of this class.[2]

[1] In particular, note that the run that calculates your code score is based on what is submitted to the assignment, and the staff solutions. These may be different than what is submitted to the leaderboard (if you choose to use the leaderboard). The leaderboard can certainly be helpful to tweak your strategy, but ultimately you will need to use *some* general principles to expect your code to work well in an environment it hasn't seen before.

[2] For example, if for some reason your solution wants to find the minimum spanning tree in a graph, you may search

Some Tips for Strategy Designs

Here are a few thoughts to help you get a sense of how we view/grade the assignments.

- **The Strategy Designs are not Programming Assignments**. For example:
  - It is possible to get full credit just through the writeup.
  - The "hard part" of the code you write is deciding what to code, rather than figuring out how to implement it.
  - The course staff will not read your code (we will just run your code).
  - The course staff may not help you debug problems that are specific to your code (we will debug issues with our own infrastructure). You can still post any quick questions on Ed. Other students in the class tend to be helpful, and the course staff will still try to be helpful, but this isn't the primary focus of the assignment.
  - All provided infrastructure is a 'bonus' to help you design strategies, *if you choose to use it, and there is no expectation that you choose to do so*. The course staff will try our best to keep the leaderboard running, but it may be down for extended periods of time.[3] When this happens, please let us know with a public post on Ed, but be prepared that it might stay down for a while. The course staff will also provide a test kit in TigerFile to help confirm that your code compiles, but we don't promise that it will catch every possible bug. If you find any issues with the provided infrastructure, please let us know. But please also think of the infrastructure as a bonus, and do not rely on it to complete the assignment — the assignments are designed so that none of this infrastructure is required.[4]
- That being said, implementing your strategy will:
  - Directly improve your score via part c.
  - Indirectly improve your score, because it will force you to have a concrete plan in mind (rather than vague ideas).
  - Feel fun to test your strategy against the course staff and your classmates (hopefully).
- Each Strategy Design should feel comparable in size to one problem on one PSet (rather than an entire PSet).[5] If you find yourself spending time comparable to an entire PSet on the Strategy Design, it may be a good idea to confirm expectations and grading norms with the course staff.
- Each Strategy Design may take a while just to understand the model (moreso than a PSet question). I suggest setting aside time to do this first (and office hours are an appropriate place to do this).

for a solution online and copy it. But you may not copy your classmate's code (although you may discuss extensively with your classmate how you plan to solve the problem, before coding on your own).

[3] Often, this is due to bugs in code submitted by your peers, or due to bugs/outages in TigerFile.

[4] A short historical note: originally, there was no provided infrastructure, other than a template with a sample strategy that correctly compiles. The assignments are designed so that strong strategies can be designed using just this template. Over the past several years, students on the course staff had fun providing extra infrastructure to make testing/design easier, which is now quite good (but the course-staff benchmark has not changed).

[5] In previous years, it was indeed one problem on one PSet. But because PSets are biweekly, it was suggested to spread out the due dates.

- Parts a and b on each Strategy Design are a warmup. I suggest doing parts a and b before thinking at all about part c.
  - Sometimes, part a and part b will ask what you would do in a simplified version. If you can figure out what to do in the simplified version, it will guide you towards what to do in the real version.
  - Sometimes, part a and part b will ask what you would do in a special case. If you can figure out what to do in a special case, you can sanity check your final strategy to confirm that it does the right thing in these special cases.
- Finally, here are some possible approaches for an effective part c:
  - Theory-heavy: one approach might make a concrete prediction about what other students will do, and then give a proof outline that your strategy is the best thing you could possibly do in response. For full credit, such a solution should give a brief argument why the concrete prediction is reasonable, and the proof should be correct (but it is OK to skip calculations and be less rigorous than a PSet).
  - Experiment-heavy: another approach might propose a parameterized framework, and then optimize parameters set using experiments. For full credit, such a solution should argue why the framework makes sense (i.e. why the framework is rich enough that *some* setting of parameters is likely to be an awesome strategy). The solution should also briefly argue why the experiments should be reasonably predictive of the student submissions.
  - Principle-heavy: another approach might be to come up with good principles behind their solution, but not necessarily prove anything, nor run experiments. For full credit, such a solution should argue why these same principles lead to a correct answer to parts a/b, and give a few brief nuggets of insight for why these principles are likely to lead to a good solution for the general case (i.e. among all the other ways to solve parts a/b, why is yours good?).
  - In all cases, it may be helpful to explicitly appeal to parts a and b. For example, if you make a prediction about what other students will do in general, does that prediction match your answer for parts a/b? If you choose an experimental framework, is your framework rich enough to contain a "correct" answer for parts a and b? In general, can you explicitly argue that your solution does something "correct" for parts a and b?
  - In all cases, you should focus your writeup on logical arguments. It is completely fine to draw intuition from practice, but your own justification needs to be based on logic rather than appealing to practice.
  - **These are all just suggestions** for how to approach part c. These assignments are, by design, extremely open-ended, and it is possible to get full credit without reading this document.

How are performance scores computed?

On each Strategy Design, your job is to optimize your own payoff. In particular, your job is to optimize your absolute payoff as a number and *not* relative to your classmates. Performance scores are awarded based on how many points you score, *not* based on your rank (it is possible for the

entire class to get full credit, for example). To be extra clear, here is the process you should have in mind for how performance is evaluated:

- Run all student strategies in a pool. Denote by X_i the reward of student i.
- For all students i and course staff strategies j, replace student i's strategy with course staff j, and run the pool. Denote by Y_i^j the reward of staff strategy j when replacing student i.
- If $X_i > Y_i^j$, then student i beats course staff strategy j.
- Student i's code score is a function of the number of course staff strategies that i beats.

In particular, observe that you're "competing" *only with strategies that you never directly play against*. That is, your code score will be optimized by just maximizing your own payoff – your strategy can't benefit by helping nor hurting others'.

The strength of your writeup justification is also evaluated based on how well it convinces the grader that your strategy will achieve high payoff.

A minor asterisk. The above scenario is how you'll be evaluated on for your writeup. However, in exchange for significant runtime savings,[6] the precise code score computation may slightly deviate from the above.

In all SDs, you are expected to justify your writeup as per the scenario described above (where you are just trying to optimize your own payoff, and never directly play against the course staff strategies you're evaluated against). In all SDs, I would strongly advise to write your code as per the scenario described above too.[7] However, if you want to know precisely how the code scores are computed, you are allowed to ask on Ed.

[6] Note that the above scenario requires a factor of ≈ 100 to do a loop for each student submission, and also a factor of ≈ 30 for each course staff submission. We may sometimes try to estimate the outcome of multiple loops at the same time.

[7] This is both because (a) I'm pretty confident that no strategy could possibly benefit, even a tiny bit, by making use of the subtle differences, and (b) in case I'm mistaken, it would still wind up being quite a significant effort for a marginal gain.