Project 1 Readme Team crystalball

Version 1 9/11/24

1	Team Name: **crystalball**
2	Team members names and netids: **Jack O'Connor (joconn25)**, **Brian Rabideau (brabideau**), **Joseph Bertram (jbertra3)**
3	Overall project attempted, with sub-projects: Project: **SAT** Sub-Projects: **Polynomial 2SAT Solver (DPLL Algorithm)**, **DumbSAT with Incremental Search + Unit Clause Optimization**
4	Overall success of the project: **Successful**
5	Approximately total time (in hours) to complete: **8**
6	Link to github repository: https://github.com/Jacko7973/TOC-Project-1-crystalball.git
7	List of included files (if you have many files of a certain type, such as test files of different sizes, list just the folder): (Add more rows as necessary). Add more rows as necessary.

File/folder Name	File Contents and Use
Code Files	
SAT_lib/_SAT_utils_crystalball.py	Contains classes for creating / manipulating SAT expressions, clauses, and literals.
SAT_lib/__init__.py	Manages library context, contains functions for testing+benchmarking solvers / loading data.
DumbSAT/DumbSAT_crystallball.py	Default, unmodified DumbSAT implementation for baseline performance comparison.
DumbSAT/MyDumbSAT_both_crystalball.py	DumbSAT upgraded with both incremental search and unit clause optimizations.
DumbSAT/MyDumbSAT_iter_crystalball.py	DumbSAT with incremental search optimization.

dpll/dpll_crystalball.py	Implementation of DPLL algorithm.
Test Files	
SAT_lib/check_SAT_plot_crystalball.ipynb	iPython Notebook for generating graphs to visualize performance.
DumbSAT/check_DumbSAT_crystalball.py	Test code for verifying correctness / benchmarking performance of DumbSAT and its improvements.
dpll/check_dpll_crystalball.py	Test code for verifying correctness / benchmarking performance of DPLL algorithm solver.
Output Files	
SAT_lib/data/data_2SAT_crystalball.cnf.csv	2SAT test cases from course canvas page.
SAT_lib/data/data_kSAT_crystalball.cnf.csv	kSAT test cases from course canvas page.
DumbSAT/data/output_DumbSAT_2SAT_data_crystalball.csv	Correctness and performance data for DumbSAT with no improvements.
DumbSAT/data/output_MyDumbSAT_both_kSAT_data_crystalball.csv	Correctness and performance data for DumbSAT with both improvements.
DumbSAT/data/output_MyDumbSAT_iter_kSAT_data_crystalball.csv	Correctness and performance data for DumbSAT with just the incremental search improvement.
dpll/data/output_dpll_2SAT_data_crystalball.csv	Correctness and performance data for DPLL algorithm on 2SAT problems.
dpll/data/output_dpll_kSAT_data_crystalball.csv	Correctness and performance data for DPLL algorithm on kSAT problems.
Plots (as needed)	

DumbSAT/plots/baseline/plot_2SAT_DumbSAT_both_crystalball.png DumbSAT/plots/both/plot_2SAT_MyDumbSAT_both_crystalball.png DumbSAT/plots/both/plot_3SAT_MyDumbSAT_both_crystalball.png DumbSAT/plots/both/plot_4SAT_MyDumbSAT_both_crystalball.png DumbSAT/plots/iter/plot_2SAT_MyDumbSAT_iter_crystalball.png DumbSAT/plots/iter/plot_3SAT_MyDumbSAT_iter_crystalball.png DumbSAT/plots/iter/plot_4SAT_MyDumbSAT_iter_crystalball.png dpll/plots/plot_2SAT_dpll_crystalball.png dpll/plots/plot_3SAT_dpll_crystalball.png dpll/plots/plot_4SAT_dpll_crystalball.png	Performance visualization of DumbSAT with both improvements (2SAT data set). There are roughly 100 data points for each SAT level, and they are each represented according to the legend markers (either satisfiable or unsatisfiable). After collecting all the data, we have studied the trend of the relationship by overlaying a regression line on the datasets.

8	Programming languages used, and associated libraries: **Languages:** - **Python** - **Jupyter Notebook** **Libraries:** - **SAT_lib (Custom built Python SAT library in the SAT_lib/ folder)**
9	Key data structures (for each sub-project): **DPLL:** - **List (SATExpression (clause list), SATClause (literal list))** - **Set (for identifying unit clause)** - **Dictionary (for assignment storage)** **DumbSAT:** - **List (wff, assignment)** - **Set (for identifying unit clause)**
10	General operation of code (for each subproject) **DPLL:** - **Implementation of the DPLL algorithm in the dpll/dpll_crystalball.py file…** - **DPLL function takes a SATExpression object as an argument (can be created with SATExpression.from_str('(a | !a) & (b | a) & …') or with SATExpression.from_list([[1, -1], [2, 1], …])), and an assignment of variables in the form of a dictionary.** - **Make a copy of the expression so it can be modified.** - **Remove redundancies in the form of literals that evaluate to false, or clauses that evaluate to true.** - **Checks for finality: No more clauses -> satisfiable, empty clause exists -> unsatisfiable.** - **Checks for unit clause. If one exists, make the correct variable**

	assignment and restart algorithm. - Finally, choose a variable and try assigning both 0 and 1 to it, restarting the algorithm each time. - **Tests implemented in dpll/check_dpll_crystalball.py** - Imports the test_solver function from our custom SAT library. - Runs the test_solver function on the dpll implementation, given an input file (DIMACS csv format) and output file location. **DumbSAT:** - **Incremental search in DumbSAT/MyDumbSAT_iter_crystalball.py:** - Takes in all the same arguments as the DumbSAT example. - Starts by creating a list of the possible input sets of each variable in the wff, each being {0, 1}. - Uses python's built-in itertools library to generate the cross product of all variable input sets one at a time. Each item in this cross product is a possible assignment to the wff. - Iterates through each variable assignment one at a time. - **Incremental search AND unit clause optimization (BOTH optimizations) in DumbSAT/MyDumbSAT_both_crystalball.py:** - Begins by generating the list of possible input sets, the same as the incremental search approach. - Searches through each clause in the wff to find unit clauses. When it finds one, removes a character from that variable's input set so only the satisfying assignment can be made. - Finally, it incrementally searches through each assignment to try to satisfy the wff, this time with a reduced search space.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. **We tested our solver using expressions involving clauses that have 2, 3 and 4 literals each. We got these files from FilesProject 1 files/SAT files/SAT Test Files on the course canvas page (in DIMACS csv format) and stored them to SAT_lib/data/data_2SAT_crystalball.cnf.csv and SAT_lib/data/data_kSAT_crystalball.cnf.csv in the project repository.** **To standardize the testing/benchmarking process, we wrote two functions in SAT_lib/__init__.py, one which processes the DIMACS format and produces SATExpression objects, and one which takes in a SAT solver function, runs each test case in the given test file, and times each one, writing the results to a csv file. This allowed us to both test the correctness and performance of our SAT solvers. Then, using the CSV output, we wrote an iPython notebook that produces graphs to visualize the performance, and saved it to SAT_lib/check_SAT_plot_crystallball.ipynb.**
12	How you managed the code development **We used GitHub to share and store our code. The reason is that every group member has access to updated and real-time code files; in addition to practice incremental development of our project by constantly doing push and pull commands. We also used Replit as our choice for a collaborative code working environment (similar to Google Colab for Python). This makes editing, commenting, and viewing the code easier for all group members as it gives**

	real-time updates to our changes.
13	Detailed discussion of results: **For DPLL, we saw an impressive improvement in performance over the naive DumbSAT approach. For the 2SAT in particular, it is clear for the data (visualized in dpll/plots/plot_2SAT_dpll_crystalball.png) that this algorithm is able to outperform the standard ‘exponential time’ solution to 2SAT, producing benchmarking results in ‘polynomial time’. 3SAT and 4SAT results (also visualized in graphs in the dpll/plots/ directory) showed significant improvements, but still appear to be ‘exponential time’.** **For DumbSAT, there is a huge improvement in performance for both implementations (iterative and unit clause) compared to the performance that uses just the iterative approach. For instance, in the case of the most complex 2-SAT dataset (the red triangle marker to the very right), running it on both implementations gives us close to 30s runtime in comparison with just the iterative approach that gives us about 65s runtime (see plot_2SAT_MyDumbSAT_both_crystallball.png, plot_2SAT_MyDumbSAT_iter_crystalball.png for reference and detail results). Both the both implementations and just iterative approach are still better in runtime and performance compared to the default DumbSAT provided (about 150s runtime) For more complex expressions like the 3-SAT and 4-SAT, the iterative approach shows a slight edge when compared to both implementations. Our guess that this happens would point to the hypothesis that 3-SAT and 4-SAT expressions do not involve unit clauses and both implementations would slightly bottlenecks the runtime since time and resources are wasted on applying the unit clause (see plot_3SAT_MyDumbSAT_both_crystallball.png, plot_3SAT_MyDumbSAT_iter_crystallball.png, plot_4SAT_MyDumbSAT_both_crystallball.png, plot_4SAT_MyDumbSAT_iter_crystallball.png for reference and detail results).**
14	How team was organized **We Split up the responsibilities of the project into a few categories, and distributed them as follows…** **- Jack: Wrote SAT_lib to make both DumbSAT and DPLL 2SAT Solver more approachable, wrote SAT solver testing procedures, assisted team members with using the library and implementing algorithms.** **- Bertram: Worked on DumbSAT improvements, generated performance graphs for data visualization.** **- Brian: Worked on DPLL algorithm implementation, worked on running test suites for testing correctness / performance.**
15	What you might do differently if you did the project again **While the goal of the SAT_lib library was to make both of the projects more readable and approachable, it added a lot of work, and introduced a slight barrier to understanding the codebase with fresh eyes. If we were to do the project again, we would modify the way we implemented SAT_lib to improve readability / ease of use.**
16	Any additional material: **N/A**