2021 AP Computer Science Programming Competition

Welcome to AP Computer Science Battle of the Schools III, otherwise known as “May Mayhem”!!!

The volunteers from IBM are excited to work with the students at Mayo High School and CTECH for a cross-school competition! The competition will determine the bragging rights among the cyber techies within the Rochester School District. The winning school receives control of the **The Belt**:

This year's project centers around the popular game **Battleship** – a common board game where pairs of players hide ships on a two-dimensional grid representing the ocean. Players shoot at their opponent’s ships by guessing the coordinates of the enemy ships. The winner of a game is the player to sink (fully cover with hits) all the opponents ships first. More information on the history of the game and how it is played can be found at http://en.wikipedia.org/wiki/Battleship_%28game%29.

For our competition, you will be writing a “computer” player for Battleship that does the guessing against the opponent's hidden board. You will be provided:

1. A runtime framework (JAR) that includes a GUI with which to run the game,
2. A DrJava project to simplify setup,
3. A sample player class file (**battleship_2021/drjava/student/player/TemplatePlayer.java**) for you to develop and enhance.
4. JavaDoc for information and framework APIs available to the player.

Scoring for the competition will be accumulated at the school level – with the winning (fastest) student earning 3 points for the school, second place receiving 2, and third, 1 point. **Given this scoring technique and the fact that most search algorithms have**

boards with which they perform well, teams should strive for diversity of algorithms within the class.

The competition will be comprised of suite of IBM-created game boards against which everyone will play. The number of guesses it takes to win the game will be the player's score. Scoring will occur for the class/school as described above.

The game will be a "classic game", comprised of a 15x15 grid with 5 ships:

- 1x5 aircraft carrier
- 1x4 battleship
- 1x3 submarine
- 1x3 cruiser
- 1x2 patrol boat

A sample number of boards will be provided to you for testing. You also can create your own boards and test against them.

In addition to the framework source code, Javadoc documentation has been written to describe the runtime classes, their hierarchy, and interfaces. A setup-guide is also provided to explain where and how to integrate your code.

Follow these steps to get started:

1. Download **student-v1.0.zip** and unzip the package.
2. In your extracted directories from step #1, locate, read, and follow the directions on how to get started in the document named **battleship_2021/doc/BattleshipDesignSetupTips.pdf**.

The goal for the competition is to provide experience with using code you have not written, while introducing the concept of Artificial Intelligence (AI) and heuristics. If one chooses to make a career as a computer programmer, you will quickly learn that the ability to read, understand, and interact with code and classes not written by you are key skills.

Good luck and have fun!!

Project Schedule

- **Wednesday, May 19**
  Competition files will be posted for students by teacher. IBM Tutors will virtually kick-off the competition.

- **During competition**
  Help will be available via Google Meet each class period. Comments and answers will be private to the school that asked.

- **Wednesday, May 26 at 4:00 PM**
  Projects must be submitted to your teacher. Late submissions will not be accepted. Your teacher will validate submission times.

- **Wednesday, May 26**
  Team of IBM Software Engineers will judge the projects.

- **Thursday, May 27**
  IBM representatives will virtually present the results.

Project Rules

1) Students will work independently and submit their own code.
2) Discussions among students on playing algorithms are STRONGLY encouraged to eliminate duplicity of solutions.
3) Students will submit one implementation of the **TemplatePlayer** class. See the detailed instructions in **BattleshipDesignSetupTips.pdf** for submission packaging.
4) Solutions must be in Java.
5) Projects will be evaluated using the criteria defined in the section below.

Project Judging Criteria

Individuals will compete for their school. Six points per board will be award to their respected schools for each board based upon the student's performance as follows:

Lowest score:	3 points
Second lowest:	2 points
Third lowest:	1 point

The three lowest scoring players will earn points for their schools. Ties will "split" the accumulated points. For example, if 3 students from 3 different schools all tie with the lowest score, then each school is awarded 2 pts (6 / 3). The school with the greatest number of points at the end of each round will be declared the winner.

Note: The evaluation will include a predefined set of (approximately 42) boards. Due to the Engineering altruism that all designs have a trade-off, one can naturally conclude that having a single algorithm which wins every time will likely be impossible. Therefore, the team which ultimately wins will likely have the most diverse set of algorithms.

Individual Awards

In addition to the priceless team award, individual awards will be provided:

1) **Top Scorer**: The individual whose algorithm scores the most points for their team.
2) **Best Team Player:** The student who solved the most unique ship placement strategy for their team.
3) **Best Program Structure**: Each student's **TemplatePlayer** class will be reviewed for organization of internal methods/functions, use of variables, use of data structures, flow of the code, and source code documentation.
4) **Most elegant solution**: Each game algorithm will be reviewed looking for the algorithm which has an innovative design that balances code complexity (quantity) and shot effectiveness (score).

Any questions about evaluation criteria should be asked during class time or communicated through your teacher. Answers for this type of question will be cross posted to all classes.

Technical Tips for a Successful Project

The first step to understanding any new programming environment is to spend time reviewing the existing code – in this case, the sample player (**TemplatePlayer**). Where does it get control? What does it do? How does it do it? These are the questions you can user to start learning about the programming environment with the sample player program installed by default.

The second step is to begin adding optimizations. In the game of battleship, there are 3 key things which can be optimized. They are:

1. Keeping track of your guesses such that you do not guess the same location multiple times. As we play a game, we want to check any guess against the list of already guessed values, especially when the guess is created randomly. Think about how you might represent the board and track your guesses. Then, ensure that your random guesses have not already been guessed. If they have, continue getting random numbers until you get a hit. Or you might consider thinking about how to apply your random number to only the remaining guesses.

2. When you get a “hit”, your next guess should be focused on sinking the ship, i.e., “localized” to the where the hit occurred. (For simplicity, let’s call this the “sink optimization”.) The challenge with this situation becomes processing all the “hits” completely in a region. One can think about this as ensuring you get a ring of misses around a hit. But what happens when you get a second hit? Techniques such as recursion OR data structures like queues or lists may be helpful in helping keep track of hits which have not been complete processed. Oh, and the guess optimization from #1 above will come in handy too.

3. Traversing the remaining free space on the board in a predictable fashion when all previous localized hits have been handled. (Let's call this the "free space optimization".) One way to explore various "free space optimization" algorithms is to think about the type of ship-hiding strategies players may use. Ships may be grouped versus spaced. They may be clustered or in a line or not connected. They may be located near one side, in the middle, or generally distributed. Think back to your strategy when playing the real game. If you knew what kind of boat distribution philosophy your opponent had, how would you search for those boats? This optimization is about translating your search strategy to a computer algorithm.

Now, let us think for a moment about team strategy. If this competition was scored individually, then most of your time should be spent trying to find an optimal algorithm for traversing all board types. However, given that this is a team competition where points accrue for the school, the class should focus on diversity of algorithms.

To explore this concept a little further, let's look closer at the various "free space optimizations" which are possible. How many ways can you "traverse" a board? Here's a starting list:

- Top-to-bottom
- Left-to-right
- Outside-to-inside
- Diagonally

Can you think of others? How about the "opposites" such as "bottom-to-top"? Don't forget those. Oh, and does the starting place matter? Yes! I can start at the top-left or the top-right sides. Will both perform the same for a given board? Nope! Likewise, how will similar algorithms perform on the same board? Near identically.

Keep all these concepts in mind when selecting your "free space optimization" algorithm. Pick one that's different than your neighbor. The winning team will have the greatest diversity of algorithms.