3. **Explain how our OOD will work**
- Pick two (2) interesting (non-trivial) actions/steps that the game must support and explain how your OOD supports this.

1. The first non-trivial action within this game is the storing of the game's grid and how each piece in the grid can take multiple states. My strategy was to have an object Cell that could act as any of the various forms a cell might have. My OOD supports this by having a separate class (Cell) that is placed into a 2d array which acts as the board. This design supports polymorphism, as multiple forms can be represented in a singular object.

2. The second non-trivial action in this game is the generation of the board, such that the polynomials are placed in a correct manner. I used a board generation algorithm that randomly chooses a start location that is not already occupied by a fort, if the polyomino can be placed the next polyomino is tried to be placed. If the polyomino can't be placed then the grid is cleared and the algorithm tries again, starting from the first polyomino. This process repeats until the correct number of forts are placed. My OOD supports this by having a private method in the Board class to generate the underlying grid. This showcases encapsulation as the implementation of how the grid is generated is hidden from the user.