|  |
| --- |
| [CS](https://www.plymouth.edu/department/computer-science/) 2381: Data Structures  (Fall 2020) |
| [**Syllabus**](https://turing.plymouth.edu/~kgb1013/?course=2381)  [**Moodle**](http://www.plymouth.edu/courses/course/view.php?id=51851)  **Teachers**  [Kyle](https://turing.plymouth.edu/~kgb1013/)  **Assignments**  [Project 0](https://turing.plymouth.edu/~kgb1013/?course=2381&project=0)  [Project 1](https://turing.plymouth.edu/~kgb1013/?course=2381&project=1)  [Project 2](https://turing.plymouth.edu/~kgb1013/?course=2381&project=2)  [Project 3](https://turing.plymouth.edu/~kgb1013/?course=2381&project=3)  [Project 4](https://turing.plymouth.edu/~kgb1013/?course=2381&project=4)  [Project 5](https://turing.plymouth.edu/~kgb1013/?course=2381&project=5)  [Project 6](https://turing.plymouth.edu/~kgb1013/?course=2381&project=6)  [Project 7](https://turing.plymouth.edu/~kgb1013/?course=2381&project=7)  [Project 8](https://turing.plymouth.edu/~kgb1013/?course=2381&project=8)  **Other Pages**  [Kyle's Teaching](https://turing.plymouth.edu/~kgb1013/?main=teaching) [Kyle's Schedule](https://turing.plymouth.edu/~kgb1013/?main=schedule) [Kyle's Resources](https://turing.plymouth.edu/~kgb1013/?main=resources) | **Project 1: Pachyderm Lists**  **Assigned: Tue Sep 01 2020 Due: 11:59:00 PM on Mon Sep 21 2020 Team Size: 1 or 2 Language: Java Out of: 100 points**  In this project, you will use good programming practice and teamwork to build a player for the game of Elephants and Rhinos. In order to do this, you'll need to use ArrayLists.  **Part 0, 0 points:**Check out the online API for the [ArrayList](http://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html) class. We'll be using this class a lot, both in this project and future projects. Create a new folder for this project and a new Java file to play around with, e.g. TestingMonkey.java where we can practice using the ArrayList methods. In order to use the ArrayList class, you need to import it at the top of your file. The API tells you which package to import: java.util.ArrayList. In order to import the package, put  import java.util.ArrayList;  at the top of your code. In a main method, create both an ArrayList of integers, and another ArrayList of strings. Practice adding and removing elements. In general, you should get comfortable with all of the following methods:   * Both versions of add * clear * contains: test both true and false cases * get, using different indices * indexOf * isEmpty * both versions of remove * set * size   **Part 1, 0 points:**In this project, you don't have any data structure to write. (I would ask you to implement the basics of the ArrayList class, but generic Java arrays are annoying to work with for some things.) Instead we're going to get directly into playing this project's game: Elephants and Rhinos. Elephants and Rhinos is a game played on a row of spaces. Each space is either empty or has one elephant or rhino in it. (Two pachyderms cannot share a space.) Each turn, a player moves one of their pieces one space forward, if possible. If not, they lose the game. Elephants, controlled by the Left player, move from left to right. Rhinos, controlled by the Right player, move from right to left. Download [ElephantsAndRhinos.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/ArrayListProject/ElephantsAndRhinos.java) to the same folder.  **Part 2, 0 points:**Let's test your code out during actual game play. You'll need some things:   * Abstract CG class: download and compile [CombinatorialGame.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/AllProjects/CombinatorialGame.java). * Player classes: [Player.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/AllProjects/Player.java) and [RandomPlayer.class](https://turing.plymouth.edu/~kgb1013/DB/2381/code/AllProjects/RandomPlayer.class). * Abstract class for generation of game states: [PositionFactory.java](https://turing.plymouth.edu/~kgb1013/DB/2381/code/AllProjects/PositionFactory.java). * Referee: [Referee.java](https://raw.githubusercontent.com/paithan/CombinatorialGameCode/master/java/Referee.java).   **Part 3, 0 points:**Add code to your testing class like the following:  int maxBoardSize = 10;  double pachydermDensity = .4; //percentage each space will hold a pachyderm  PositionFactory<ElephantsAndRhinos> factory = new  ElephantsAndRhinos.PositionBuilder(maxBoardSize, pachydermDensity);  Player<ElephantsAndRhinos> random = new RandomPlayer<ElephantsAndRhinos>();  Referee<ElephantsAndRhinos> ref = new  Referee<ElephantsAndRhinos>(random, random, factory);  ref.call();  Run this code a few times unti you watch a couple of meaningful games to see how it's played.  **Part 4, 10 points:**Now it's time to write your own player! Create a new source file, ElephantsAndRhinosPlayer.java. Just like last time, you need to tell the compiler that you're going to extend the Player class for the appropriate game. Set up your Javadoc and course header in the following way:  /\*\*  \* Fill this in later with a description of your player's strategy.  \*  \* @author Put your names here.  \*/  public class ElephantsAndRhinosPlayer extends Player<ElephantsAndRhinos> {  **Part 5, 10 points:**Add a constructor that takes no parameters. As before, it's completely fine to have your constructor be empty.  **Part 6, 10 points:**Now add a toString method that returns a String name for your player that no one else will choose.  **Part 7, 0 points:**In order to get this class to compile, we need to implement the getMove method that returns a new ElephantsAndRhinos position. We'll start off by adding a stub so that it will compile and we'll fix it later. The header to this is similar to last project's, except that you'll be using ElephantsAndRhinos as the game. Here's a template for the method body that will compile:  ... {  ArrayList<String> spaces = position.getSpaces();  //put your code to modify spaces here  return new ElephantsAndRhinos(spaces);  }  Compile all your code to make sure that everything's working.  **Part 8, 5 points:**Let's write some static methods to make sure we're comfortable with Java ArrayLists. Add a new method with the following Javadoc and signature:  /\*\*  \* Returns whether the given board has an elephant at the given index.  \*  \* @param position The game state that is being checked.  \* @param index The index of the space on the board, starting from 0 and going left-to-right.  \* @return True if there is an elephant in the indexeth space of position; false otherwise.  \*/  public static boolean hasElephantAt(ElephantsAndRhinos position, int index) {  In order to write this method, you will need to first get a hold of the underlying board from position. If you use Javadoc to produce the API page for this, you'll see all the public ElephantsAndRhinos methods:  > javadoc ElephantsAndRhinos.java  Then open the new ElephantsAndRhinos.html file in a web browser. If you scroll down, you should see the getSpaces method. You can use this by including this as the first line of your new method:  ArrayList<String> spaces = position.getSpaces();  Every element in the list should be one of only three values: one string for empty, one for an elephant, and one for a rhino. I don't need to remember what the actual values for these are because I set up three constants in ElephantsAndRhinos: EMPTY, ELEPHANT, and RHINO. For example, if I wanted to know whether the seveneth cell of a list called someList held a rhino, I could use this:  String sevenethElement = someList.get(7);  boolean hasRhino = sevenethElement.equals(ElephantsAndRhinos.RHINO);  Use this to finish writing the hasElephantAt method.  **Part 9, 0 points:**Make sure your hasElephantAt works correctly. You can create a new Elephants and Rhinos position by doing something like this:  ArrayList<String> pachyderms = new ArrayList<String>();  pachyderms.add(ElephantsAndRhinos.EMPTY);  pachyderms.add(ElephantsAndRhinos.ELEPHANT);  pachyderms.add(ElephantsAndRhinos.RHINO);  pachyderms.add(ElephantsAndRhinos.ELEPHANT);  ElephantsAndRhinos game = new ElephantsAndRhinos(pachyderms);  Then the following should be true:  boolean hasElephant = ElephantsAndRhinosPlayer.hasElephantAt(game, 3);  Add a test like this to either your player's main method or inside your testing class. Switch to the framework I showed you in the zeroeth project, then add more tests so that you're certain your method works.  **Part 10, 0 points:**Now add a hasRhinoAt static method that does a similar thing.  **Part 11, 5 points:**Add a static hasPachydermAt method that returns true exactly when there is either an elephant or rhino in the specified space. You can write this very easily by invoking both of the other methods you just wrote. Add a bunch of tests so that it's clear it works.  **Part 12, 10 points:**Let's add another static method to help us out:  public static boolean pachydermCanMove(ElephantsAndRhinos position, int index)  This method should return whether the pachyderm at the given location can move. (It should never be called on when the index points to an empty space. You can softly enforce this by adding a precondition to the Javadoc tag.)  \* @param index The index of the space. Precondition: the indexeth space must contain either an elephant or a rhino.  It may make your code cleaner to first write two helper methods: elephantCanMove and rhinoCanMove.  **Part 13, 10 points:**Alright, one more pair of these practice methods:  public static ArrayList<Integer> moveableElephants(ElephantsAndRhinos position)  This method should return an ArrayList of all indices of spaces in position that have an elephant that can move.  **Part 14, 0 points:**Do the same thing for moveableRhinos. If you want, you can also include moveablePachyderms.  **Part 15, 0 points:**Update your getMove method. Here different players move different pachyderms, so you'll need to do different things depending on whether the current player moves Elephants (Left) or Rhinos(Right). The referee tells you which player's turn it is by specifying the value of the playerId parameter. You can use that by testing against some constants I set up in the CombinatorialGame class. Here's an option for a template:  ... {  ArrayList<String> spaces = position.getSpaces();  if (playerId == CombinatorialGame.LEFT) {  //modify spaces to move an elephant  } else if (playerId == CombinatorialGame.RIGHT) {  //modify spaces to move a rhino  } else {  //this case should never happen!  System.out.println("Error!");  }  return new ElephantsAndRhinos(spaces);  }  Each element in the array-list of strings can have one of three values, which I've referenced using constants:   * ElephantsAndRhinos.EMPTY * ElephantsAndRhinos.RHINO * ElephantsAndRhinos.ELEPHANT   You can check out the code to see how I did this. (Notice I used Javadoc to describe the constants; if you build the Javadoc, you can see the entry for each of them.) The reason for using constants here is that if I decide that I want to change the values for the elements, I can just change those constants instead of changing where I used them in all my other code! You can use my template and the other methods you've already written to move the first Elephant you see on their turn:  ...  if (playerId == CombinatorialGame.LEFT) {  //move the first elephant  ArrayList<Integer> elephants = moveableElephants(position);  int first = elephants.get(0);  spaces.set(first, ElephantsAndRhinos.EMPTY);  spaces.set(first+1, ElephantsAndRhinos.ELEPHANT);  }  ...  **Part 16, 20 points:**Modify your code so that getMove always returns a legal move, *for both players*. I usually start by making the easiest move I can think of. You can test your code by adding it to the referee in your testing class. I recommend running a gauntlet of lots of runs and making sure you don't have any forfeits. To test your player in both the Left and Right roles, add it to "both sides" of the referee:  Player<ElephantsAndRhinos> myPlayer = new ElephantsAndRhinosPlayer();  Referee<ElephantsAndRhinos> ref = new Referee<ElephantsAndRhinos>(myPlayer, myPlayer, factory);  Remember:   * Always make legal moves * Don't use randomness * Don't use the getOptions() method.   **Part 17, 0 points:**Spend some time playing the game to see if you can come up with good strategies. How do you choose which pachyderm to move? Feel free to discuss with people outside of your group. (But don't share your source code.)  **Part 18, 0 points:**Translate your ideas into code. The random player actually does really well on small boards, so it's hard to see an improvement. I used a maximum board size of 1000 and a density of .8 when I was first running trials. (I won't make any promises about how I'm actually going to test it.)  **Part 19, 20 points:**Improve your player so that it consistently beats the random player:   * 15% of the time or more: 10 points * 25+%: 15 points * 35+%: 20 points * 45+%: 25 points (5 are bonus) * 60+%: 30 points   Your algorithm should work equally well for both the Left and Right player. Make sure to test it against my player both ways:  int numTrials = 1000;  ref = new Referee<ElephantsAndRhinos>(myPlayer, random, factory);  ref.gauntlet(numTrials);  and  ref = new Referee<ElephantsAndRhinos>(random, myPlayer, factory);  ref.gauntlet(numTrials);  I'll take the lower of the percentages that I see. (I may run more tests than I specify here.)  **Part 20, 10 points (Bonus):**I wrote a perfect player for this game. Download [ElephantsAndRhinosPerfectPlayer.class](https://turing.plymouth.edu/~kgb1013/DB/2381/code/ArrayListProject/ElephantsAndRhinosPerfectPlayer.class). Test it against your player. Improve your player so that it consistently beats my perfect player (in both roles) over 49% of the time. If you do this, congratulations! You've probably written a perfect player! Great job!  **Submitting your Project:**  Be careful to follow all these directions precisely in order to ensure I can grade your assignment in a timely manner. If you don't, I may give you zero points for the assignment.  Make sure your code all compiles from the command line:  javac \*.java  Make sure your player class doesn't use any methods of the data structure except for the ones I asked you to write. (I will be testing your player with my own version of the data structure. That way if there are any issues with your data structure you didn't notice, your player can still run correctly.)  Please check that your code only prints to the screen in the tests and not when being tested by the Referee. If it does, please comment those print statements out.  Create a *new* folder to submit your files in.   * If you are working alone, Name this folder *YourLastName*Project1, all in PascalCase. (For example, my folder name would be: BurkeProject1.) * If you are working in a group with two people, create a new folder labelled with your two last names, separated by "And", and followed by Project1 all in PascalCase. (For example, if I worked with Kathleen Stock, our folder would be named BurkeAndStockProject1.)   Copy your source code (.java files) into that directory. Your code should be at the top level of that directory (not in any subdirectory). You do not need to submit any extra classes you used to test your structure or player. (I've got really tough tests to do that myself!) I'll be looking for the following files in your folder:   * ElephantsAndRhinosPlayer.java   Finally, compress your file into a .zip file. (Don't use WinRar!) Upload the zipped file to the project page on Moodle. If you're submitting late, please also send me an email (without the code attached) so I know there's something waiting for me on Moodle. |