Note: this document may grow with the answers to

1. You should already know how to create a class. Note that you may want to tell your IDE to create a main() function for you too when you create the class (see point 2).

2. You have seen me create instances of classes in main and call a method on these straight away in lectures a number of times.

3. The hard bit here is remembering to use these constants rather than hardcoded numbers throughout your program – that’s why I put this near the start of the requirement list. Declare normal member variables and add the keyword final (just like you would add static to make a variable static). This makes it a constant, so you can never change its value after it has been created. This means that you must set its value at the time it is created.

4. You need this function because your buttons will call it later (step 10). Providing it now (even though not implemented) means that you can concentrate on getting the colour button right first.

5. Look at how ColorLabel was implemented for this, since it is so similar. If your main class was called PszjaMain then an object reference of this class would be something like “PszjaMain mainClassObject”. You should have a parameter of this type on the constructor and a member variable of this type in the button objects. The constructor should store the value of the parameter into the member variable, so that it is available if it is needed later. i.e. if the button needs to use the main class object (see step 10) then it has this reference to the object which it can use.

6. I used two variables per button for the state – one for whether the button had been finished (was black) and one for whether it was showing at the moment or not (i.e. grey or coloured). You could do it in one variable per button though if you wish.

7. The idea is that when you select a button it does the equivalent of flipping a card over to see what colour is on the other side. i.e. selecting it will reveal the colour hidden in it. Afterwards, if the buttons were not matched it flips them back again, showing the grey back again. This should be easy to do – it’s like a setter method but will need to set the values of the attributes appropriately. If you wish you can prevent a finished button from being flipped, but it will depend upon how you implement the button.

8. The button needs to know the draw colour and your main code will set these – see steps 18 to 20.

9. If the button is finished it should be black regardless of whether the button is flipped or not.

10. Remember that the reference to the main object and the button number are both stored in attributes of each button – see what you did with the constructor in step 5.

11. This is not meant to be hard to do – see step 2 as well and make sure it links up with what you did in step 2.

12. To get the colour to appear you may need to tell your labels that they are opaque rather than the default of being transparent. The function to do that should be obvious. You have seen setting a font before.

13. Don’t make these variables static!!! If you do then you won’t be able to have two instances of your game running within the same program at once.

14. This should show whose turn it is, what the score is, and what the current player should be doing.

15. See step 12. The point here is that the colours will change over time as the game progresses – I.e. The red and green will switch for the players.

16. This means making sure that the score are shown correctly.

17. Note the use of the constants.

18. This can be tricky to do. There are a lot of ways to do this. Personally I kept a track of how many buttons I had created of each colour, then randomly chose a colour for the button and chose a new colour if the maximum number of buttons of that colour had already been created. We won’t ask you to run it with more than 8 colours. There are a number of different standard colours you could use, e.g. Color.RED, Color.BLUE, etc. You could use a switch statement to determine a colour, or create an array of colour object references, or The Random class is a good way to create random numbers. Only create one of these objects though, rather than a lot of these.

19. See step 18

20. See step 18

21. Think about the flip() method on the button, the state variables that you need and/or have already implemented. Consider steps 12-15.

22. Very similar to what you did in step 21, although the players can’t choose the same step twice.

23. Much the same as step 21 but this has to do the work of working out the results of the click.

24. Note that some things only happen after the third click, even though you worked out what would happen in step 23.

25. This tests you ability to correctly set the state back to the beginning state. You may find it useful to set the original state and to reset the state now I. The same function and just call it from both places.

26. This involves some state changes again.

27. You can either check the state of each square or just look at how many points have been scored, or any other method you can think of which will always work. To display the message window, look up JOptionPane. It’s a good idea to get used to looking up how classes work, and giving you the class name to look for you will find loads of examples on the web. I used the showMessageDialog(frame,...) static method in my code. Note that making this function call will mean that the lines of code after it will not be executed until the window closes – which is probably exactly what you want.

28. This requirement is all about putting the state back to what it was originally.

29. This mark is for implementing the Iterable pattern to create an appropriate Iterator instance. As long as you can demonstrate your code using it somewhere, you don’t need to have the use of the iterator as a fundamental part of the game. If you do want to actually use it rather than just putting in some test code to demonstrate, then either checking for all buttons being black, or in the assignment of random colours to buttons are two places where you could use the Iterator.

30. This mark is for your iterator instance working correctly to iterate through the objects. You should be able to demonstrate this to the marker.