

PPA Assignment 13

Overview

Do you think there is a problem with any of the content below? Let us know immediately at programming@kcl.ac.uk.

Read through this brief carefully before starting your attempted solution. Also ensure that you comment your code.

You are not advised to print this assignment, but instead to always access the latest version of this brief through KEATS, which will evolve with minor clarifications and corrections throughout the assessment period. Students will be notified of any major modifications to the brief by email.

A partner from your lab session should be selected for this piece of work at the next available opportunity, typically your next lab session. You must not complete any of the assignment below without your chosen partner present, as doing so is likely to jeopardise your grade.

You must select a new partner for this piece of work, to ensure that there is no impact on your mark.

Aims

The aims of this piece of coursework are as follows:

Once completed, both you and your partner must submit your assignment using the link marked 'Assignment 13: Nexus Submission Link' on KEATS, which will become available after the first pair programming lab session. However, this is not enough to receive a mark for this piece of work. You must also attend the lab session following your submission, so that one of the teaching assistants can mark your work with you present, and ask you detailed questions about it. Revisit the 'Lab Assessment and Pair Programming Q&A' guide on KEATS for more information.

Any submitted code or documentation that is found to be unduly similar to the code or documentation submitted by any other pair(s) of students, will result in a penalty for those involved.

Provisional marks for your code will be released on KEATS within one week of the final lab assessment. Final assignment grades will be submitted to the exam board at the end of the semester.

- To better understand and critique the model-view-controller (MVC) paradigm.
- To practise further with the use of widgets (components) and event handlers.
- To practise using the Java API, and associated resources, to work with new UI concepts, such as the use of images.

As this assignment is designed to test the knowledge acquired over a two week period, you may not be able to complete all of this assignment until after the PPA lecture following its release, where further language concepts will be presented to you.

In addition, as with many of the tasks this semester, and as part of your development as programmers, you will be expected to conduct some additional research, using the Java API, in order to complete some parts of this assignment.

Preliminary Tasks

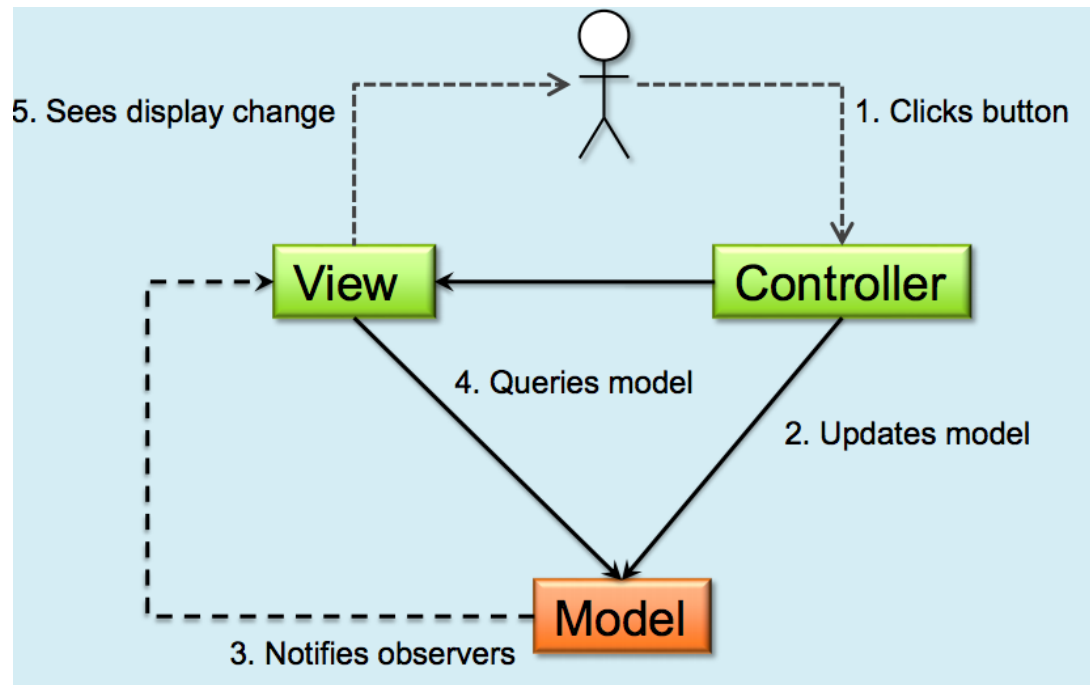
In CW12, you were asked to implement a flight simulator, and in doing so consider how best to structure such a program.

You probably considered a natural structure, in which you developed a set of classes -- like the ones you implemented in the first semester of PPA -- to represent the key entities in your program (e.g. the plane, the runway), and a class, or set of classes, to represent the new visual elements of your program, which you are learning this semester.

For all other queries, see the Support section on KEATS, specifically the 'Lab Assessment and Pair Programming Q&A' guide.

Following Lecture 5, you should now be aware that this natural structure is formalised and developed by the model-view-controller paradigm.

Therefore, as a preliminary task for CW13, you are asked to review the structure you chose for CW12, and if necessary, restructure it so that the communication between your classes fits the MVC structure given on Slide 18 of Lecture 5.



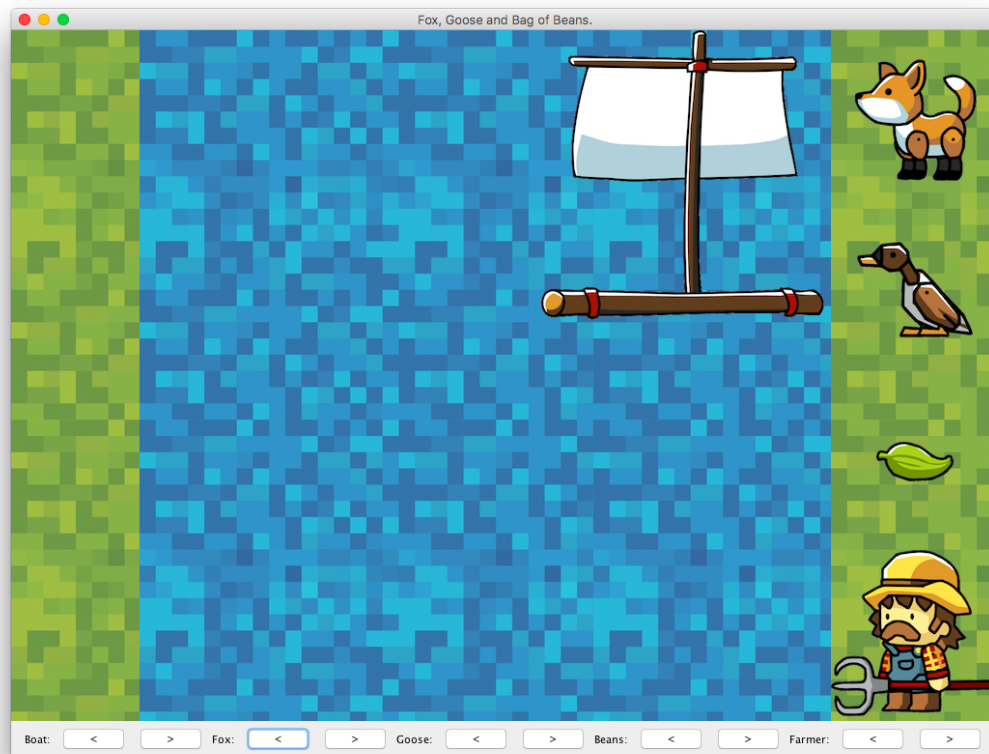
If you feel that your submission for CW12 already matches the structure shown on this slide, you should add additional comments to your code in order to highlight where each element of the diagram shown on Slide 18 is implemented in your code.

The classes you create or update for this task should be stored in a package called ``preliminaries'` in your ``assignment13'` project

Task Overview

For the main part of this piece of coursework, you will build a program to allow the [Fox, Goose and Bag of Beans](#) puzzle to be played and solved. You are encouraged to read up on this problem, if you are not already familiar with it, before proceeding.

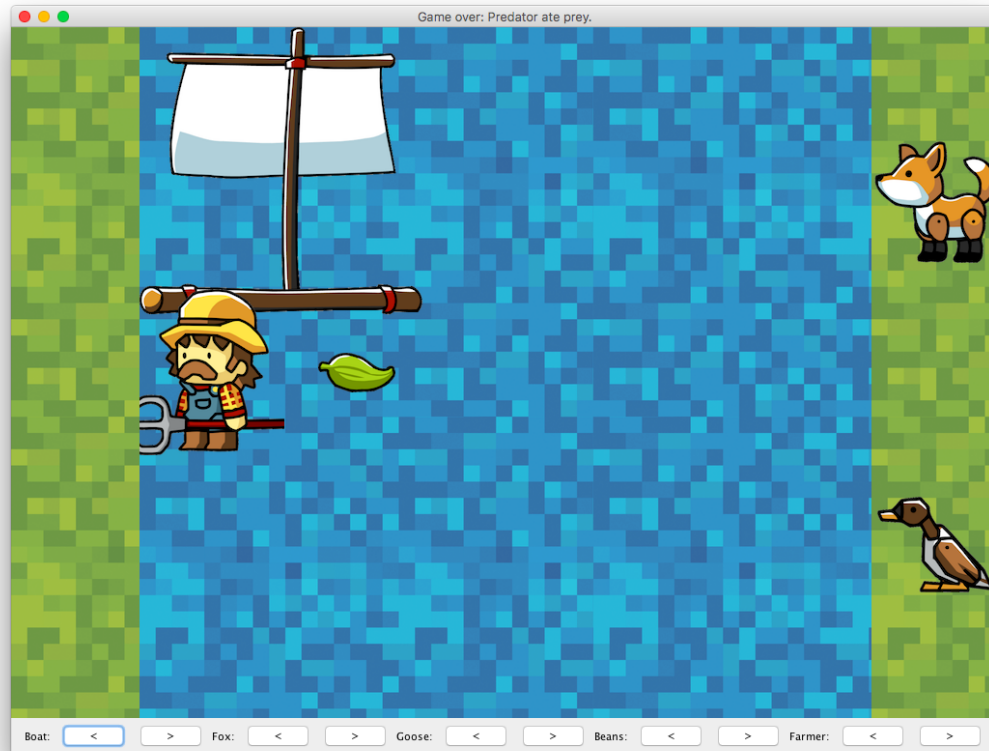
A proposed UI for this task is as follows:



Based on this UI, and the problem, an overview of the functionality required by your program is as follows:

1. The UI should be separated into distinct sections showing the two sides of the river, and the river in the middle.
2. Appropriate icons should be placed on the right hand side of the river (where the farmer and his purchased products start, hereinafter called pieces), and a boat icon should be placed in the river.
3. Labelled buttons should appear along the bottom of the screen, enabling the user to move each piece either left or right. Naturally, if a piece cannot move left or right because they are, for example, at the edge of the board, then nothing should happen when this button is clicked. Alternative mechanisms for interacting with the pieces on the board are acceptable, so long as the functionality is the same.
4. It should only be possible to move the boat when the farmer is on it.
5. The player should be scored, such that every time the boat moves, 1 is subtracted from their score, which starts at 0. Therefore, players who make the least moves will have the highest scores, although these scores will always be negative. This information should be shown in the title of the main frame, replacing the text shown in the image above.
6. There should only be room for two pieces on the boat at the same time, which must always include the farmer if the boat is to move.
7. A piece should only be allowed to move into the boat, if the boat is currently adjacent to the side of the river upon which that piece currently exists.
8. If, at any point, a fail state is reached in the game (e.g. the fox is left alone with the goose, or the goose with the beans), then the frame title text should be

replaced in order to inform the user of this, and it should not be possible for the user to continue using the controls to navigate through the game.

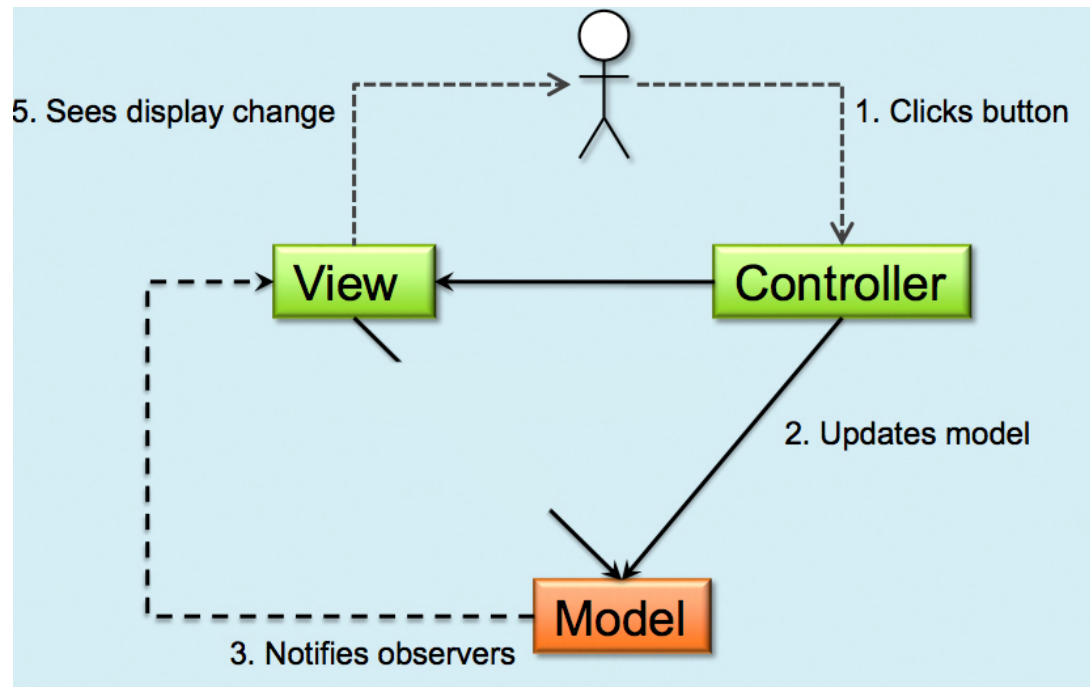


9. Once all the pieces, including the farmer, are on the left hand side of the river, the game should end with a message of success being displayed in the title of the main frame.

Considerations

1. For this task, you are required to structure your code around a more rigid form of the model-view-controller paradigm, which is shown below. This form of the

paradigm still maintains the fundamental separation between the model and the view, but restricts the view from directly referencing the model, forcing us to think more explicitly about the role each of our classes plays. During your implementation, consider the advantages and disadvantages of this form of the paradigm, when compared to the structure you produced for the preliminary task. Your examiner may ask you about this.



2. In order to ensure that this rigid form of the model-view-controller paradigm is adhered to for the assessed task in this assignment, you are instructed not to include any references to any of your model classes from your view class(es) and no references to any of your view class(es) from your model classes. You can test this by temporarily moving your model classes, and seeing whether your view class(es) compile without error, and vice-versa. Your examiner will check your code for any of these references during your assessment.

3. In addition, you should consider whether you have the correct division between the `view` element of your program and the `model` element of your program, by asking the following: would your program still provide the required functionality, if any classes containing UI code were removed, and replaced with a simple main method that makes direct calls to methods, and prints results, like we saw last term? If not, you may need to separate your code further.
4. Similarly, another good heuristic for having the correct separation between the model and view elements of your program, is to only have fields and parameters in your UI class that are either of a primitive type, or of a Swing or AWT type, or an extension thereof.
5. In your model, you should aim to create the richest and most accurate representation of the entities in the domain as you can. This should include logical inheritance hierarchies, with no common parent classes where this doesn't make sense conceptually. You are advised to make use of interfaces to help you with this.
6. In addition, we want to make our code as extensible as possible, and account for the introduction of similar types of pieces in the future (e.g. animals with similar traits to foxes and geese). Again, the use of interfaces may help you obtain this extensibility.
7. To add images to your view, you may like to look into the capabilities offered by the [JLabel](#) class, the [BufferedImage](#) class, and the capabilities offered by the [JComponent](#) class, which the JPanel class extends.

Optional tasks

1. When a fail state is reached, you may like to indicate why this is using the images on the UI themselves, such as a graphic to indicate who has eaten what.

You should also capture this in your model, and ensure that you do so with extensibility in mind, as mentioned above.

2. Once you have solved the game yourself, make a start on writing an AI to solve the game for you. Which techniques can you use for doing this?

Attempting this task will also be a good test of whether the model is set up correctly, and captures all the interactions that the UI does, as a non-human player should be able to interact directly with the model classes in your program, and have access to discreet state information.