Design & Diagrams

Documenting your designs

You are expected to be able to produce ***UML class diagrams*** and ***UML interaction diagrams*** (i.e. sequence diagrams and/or collaboration diagrams). Class diagrams were presented during the live coding sessions in the first weeks of the semester. Interaction diagrams are covered in lectures too. You have been provided with a set of class diagrams that cover the engine code and basic game code that we have provided to you.

Your design documents may be created with any tool that you choose — including a pen and paper if you can do so legibly. However, **you must create PDF, JPEG or PNG images of your design documents in your Git repository** (also include your original source files, so that you can share them with your partner and so that you’ve got a backup of them). We cannot guarantee that your marker will have the same tools available that you used (or the same versions).

If you want a UML diagramming tool, there are a few suggested tools under “Assignment Resources” in Moodle (see Assignments).

*As you work* on your design, you must store your design diagrams in your Git repository. This will help your marker verify that you have been completing your share of the work on time as agreed.

Class diagrams

We expect that you will produce *class diagrams* that cover new or changed classes that implement new features. Your class diagrams do not have to show the entire system. You only need to show the new parts, the parts that you expect to change, and enough information to let readers know where your new classes fit into the existing system.

As it is likely that the precise names and signatures of methods will be refactored during development, you do not have to put them in this class diagram. However, **the overall responsibilities of the class need to be documented *somewhere*,** as you will need this information to be able to begin implementation. This can be done in a separate text document, or you can describe the responsibilities of each class in its classbox in the class diagram as recommended by Fowler in *UML Distilled* (3rd edition).

When you are drawing your class diagram, please bear in mind that it has a purpose: it’s there to help your marker understand your system, and to evaluate the quality of your design. Your marker will want to know:

* **what things you have chosen to model using classes, and how those classes interact**
* **whether you have considered how desired functionality can be implemented**
* **what the roles and responsibilities of any new or significantly modified classes are**
* **how these classes relate to and interact with the existing system**
* **whether you are using new object-oriented techniques such as inheritance in an appropriate way**
* **whether your classes, methods, and packages are too large or small**
* **whether you are trying to minimize dependencies between classes, methods, and packages.**

You don’t need to put all of your classes into one class diagram. In fact, doing so would probably make the diagram very hard to read. Instead, consider drawing separate class diagrams for each package. You should show all classes, dependencies, and associations within the package, but the only classes in other packages that need to be shown are those that your classes depend on. See the game package documentation for an example of this style of documentation.

Interaction diagrams

You should use *interaction diagrams* (e.g. sequence or collaboration diagrams) to show how objects interact within your system. These are only needed for complex interactions. As a rule of thumb, a “complex interaction” is one which:

* involves at least three interacting classes, and
* involves more than one class sending messages.

If you are unsure if an interaction is complex enough to require an explanatory diagram, consult your marker.

Your interaction diagrams should show:

* all objects involved in the interaction, and their classes
* all messages sent (i.e. methods called)
* values returned from messages, if not trivial

Scope of an interaction diagram

In other units, especially in domains such as business process analysis, it is common for an interaction diagram to be drawn for a *use case*. The UML itself does not limit the use of diagrams in this way. For the purposes of this unit, use-case based documentation probably involve a lot of boilerplate documentation showing the user selecting menu selections. For that reason, **we suggest that you limit the scope of your interaction diagrams to showing the interactions behind one or two key method calls each** – you can (and should!) place a note or label each diagram to make its scope clear.

**Hint**: Good candidates for interaction diagrams often include complex playTurn() or Behaviour operations, which need to take many factors into account when choosing which Action an Actor will take on the next turn. However, you can diagram out any operation that you feel is complicated enough to be explained in detail.

REQ5:

# **Scenario**

The coin($) is the currency that player uses to trade with items. A coin has an integer value that determines the actual value of the money. Coins will spawn randomly from the Sapling (t). The collected coins can be traded with Toad for the following items:

1. [Wrench](https://edstem.org/au/courses/7092/lessons/17476/slides/123928): $200
2. [Super Mushroom](https://edstem.org/au/courses/7092/lessons/17476/slides/127092): $400
3. [Power Star](https://edstem.org/au/courses/7092/lessons/17476/slides/127092): $600

Toad (B) is a friendly actor, and he stands in the middle of the map (surrounded by brick Walls).

It is recommended that you have a wallet system that can store the total balance of your money.

# **Implementation Expectations (A2)**

* You should see multiple actions to buy an item

Graphical user interface, text, application, chat or text message

Description automatically generated

* When you buy Power Star or Super Mushroom, you cannot drop these items.
* You can consume Super Mushroom anytime, but not for Power Star (which has fading duration, including when it is inside the inventory).
* If you don't have sufficient money, it must print "You don't have enough coins!" and so you cannot get the item.