Puckemon

System Design Document

bEsTpRoGrAmMerSeU

Rasmus Almryd, Lukas Jigberg,
Emil Jonsson, André Kejovaara

Objektorienterat programmeringsprojekt - TDA 367

Chalmers tekniska högskola

Oktober 2021

Contents

1 Introduction

This document aims to present the game application Puckemon. The document also presents the game system architecture and design as well as how the application handles data management and the quality of the code.

1.1 Definitions, acronyms, and abbreviations

- **Puckemon**: A monster that is used to fight other monsters
- **(Player bag)/(Trainer bag)**: A bag containing Puckemons
- **HP**: Health points, the amount of health.
- **Wild Puckemon**: A single Puckemon meant to function as a single opponent (not part of a Trainer bag).
- **Combat**: The Player using its own team of Puckemon fights against an opposing Wild Puckemon or Trainer. Combat is ended when the player or opponents Puckemon all have fainted.
- **Experience points**: Points which at a certain amount is supposed to level a Puckemon up, which in turn increases certain stats which makes the Puckemon stronger in combat.
- **PMD**: Short for Programming Mistake Detector, which analyzes the source code and reports any issues found
- **GUI**: Graphical User Interface, which refers to what the user sees when using the application.
- **Repo**: Short for repository, a location to store software packages.
- **JUnit**: Unit testing framework
- **Viscosity**: This refers to how easy it is to add code to the program while maintaining the design.
- **MVC**: Refers to a design pattern named Model View Controller. **application**: An application that runs locally on the device and does not require anything else to be functional.
- **PP**: Power points, the amount of times left to use an attack.
- **Fainted**: Describes a Puckemon which has been defeated.
- **Party**: A collection of Puckemon.

2 System architecture

Puckemon is a standalone application that only requires the system to have Java installed to function properly. The application contains all the game logic within the application itself. The application uses an xlsx-file to store data.

The ExcelReader class uses the dependency POI (see [2]) to read the excel file. LibGDX (See [1]) is used for the visual part of the game. These dependencies have been implemented with maven (See [3]).

2.1 Application cycle

The application starts with the main screen that welcomes the user to the game along with some short instructions for how to interact with the game. The welcome screen also gives them the option to start the game which loads the combat screen. This screen contains the players primary puckemon and the enemy puckemon. The enemy puckemon can either be in possession of another puckemon trainer, or it can be a single wild puckemon. A puckemon trainer can have multiple puckemons in their bag, the same way a player can have multiple puckemons in their bag.

The flow of the combat is turn-based, where the player is presented with four different main approaches: Attack, Inventory, Switch and flee. The enemy trainer will make one of three choices; attack, switch or pick an item from the inventory. The wild puckemon will only be able to choose an attack at random. The choice is then executed according to this priority list, where 1 is highest:

1. Switch
2. Items
3. Attack, where the attack with highest attack priority is prioritised.
4. The Puckemons speed attribute, where a higher value is prioritised.

After the round is complete another turn is started if both Puckemon has HP left.

The combat is over when either all the player puckemons have fainted or all the opposing puckemons have fainted. The player will receive experience when the combat is won and prompted to choose from three different options to continue a new battle depending on which difficulty the user wants.

3 System Design

The UML diagrams of the application can be found under Appendix.

3.1 System Packages

This section includes some short descriptions about what the packages do and how they interact with each other.

3.1.1 Model

The relationships of the packages in Model (See 11) have one sort of circle dependency which is by design. The Puckemons have attacks, which have effects, which manipulate Puckemons. This is because one of the early decisions of the application was that certain attacks or items, which both have effects, would be able to modify the fighting Puckemons in a lot of ways. This was decided so the range of what the effects could do with the Puckemons were very wide. But it is very important to realize that the effects have a dependency on IPuckemon, which the OwnedPuckemon or FixedPuckemon classes that have the effects are implementing. So the effect only calls methods from the interface to manipulate the actual implementations of either OwnedPuckemon or FixedPuckemon. This also allows more methods to be added to the interface should the need come up.

The Combat package then executes these effects when a player asks to use an attack or item without knowing the underlying implementation of the effects. This is done through the IEffect method (See 17) "execute". By doing this the implementation of the effect is hidden.

3.1.2 View

The view (See 6) is mainly made up of screens that are used in the run package to display the different parts of the game. The animation, menu, screenObjects are then used in the screen to be displayed. Animation are made up of classes that are meant to be rendered and change over time, giving the user the illusion of movement. The Menu package (See 7) contains the classes that are used display and interact with different types of menus. The menu package also works together with the controller package (See 3) to make changes to the model when different menu options are pressed. To loosen coupling the menus use observers to notify the controllers. The menu package also utilizes screenObjects to display different types of menu options in an modular way.

3.1.3 Controller

The controller package is just a singular package containing the different controllers used by the game and menus. The controller package closely works together with the model package and its sub packages, to be able to change different model states or execute some action. Some controllers also have access to the view package to change menus and/or the run package to change screen and controller for the game.

3.1.4 Run

The run package has the task of starting the game and connect the view, controller and model. Due to the way the game library LibGdx is set up, some double dependencies is made with view and controller. We have reduced the problem by implementing an interface to loosen coupling.

3.2 MVC

The application uses the MVC design pattern to present the view to the user, get input from the user and handle the model of the application (See 2). The controller package updates the model and view package depending on the input form the user. The view retrieves information from the model and displays it to the user. The view package also have some classes that updates the

controller with menu events through an observer. View uses the serviceController (See 4) to access the MessageObserver and EffectObserver.

3.3 Domain model

The domain model (See 1) consists of the "Project Puckemon", which is supposed to describe the main combat of the game. The trainer and player of the domain model has, just like the design model, bags with puckemons in them as well as inventories with items in them. This is clearly mirrored in the actual game where the player and opposing trainer has inventory and bags with puckemon. Puckemon, also like the game and design model, has attacks. There is also instances of wild puckemon, which just like the game and the design model is a puckemon and therefore also has attacks.

3.4 Abstraction

The application is developed to depend on abstraction and not details in order to satisfy the dependency inversion principle. The application uses screenobjects to render objects on the screen which allows for better code reuse and it's easy to extend the application if one wishes to implement more graphical objects.
The application also depends on interfaces rather than classes in many instances. Some examples of this are the interfaces IPuckemon, IEffect and IFighter. The abstract class Puckemon which uses a hierarchy implementation also helps making the code more abstract and reusable.
The model uses the ServiceControllers to work with the controller CreatePuckemon to use the service ExcelReader. This approach makes the process of creating Puckemon very easy and hides the implementation of the ExcelReader service.

3.5 Foolproof parameters

Due to the large amount of unique variables that need changing in the code was it crucial to use something more reliable then setters. The solution were to instead use public methods that decrease or increase values. Reducing the risk of errors by not allowing incorrect types to be parsed.

3.6 Observer pattern

The observer pattern is used on a few occasions. Observing when new messages are supposed to be printed and when to display effects on the combat screen. Having an observer design pattern in the application helps to keep the code low coupled.

3.7 The Single Responsibility Principle

The application's packages and classes are designed to keep a single task as focus which aids the process of having a modular application. The modularity of the application makes the process of adding and removing functionality without affecting other classes easy. Most software needs regular maintenance, therefore following the single responsibility principle gives a low viscosity to keep the code easy to read and maintain.

4 Persistant Data Managment

There are mainly two types of data which are stored, sprites for Puckemons and the data describing each Puckemon. The sprites, or images, of each Puckemon is stored under the "src/main/resources/back" or "src/main/resources/front" folders. The Puckemon sprites are named x.png where x is the ID of the Puckemon. These sprites have a back and front version each stored under the "back" and "front" maps respectively, with the same ID.

The data describing each Puckemon is stored in a spreadsheet under "src/main/resources", which is used to construct the Puckemon and give it stats. Here the ID of the Puckemon is also used to identify which Puckemon is referenced in the spreadsheet.

Apart from this there are images for items under the "src/main/resources/items" folder, which are important to know. There are also backgrounds and other ui items stored under the "src/main/resources" path and some subfolders which are used throughout the application.

5 Quality

5.1 Testing

The application has been tested using JUnit and github actions for continuous testing. Testing of the model has a class coverage of 100 percent and a line coverage of 95 percent. There is also testing for the "services/puckemonGenerator" package, which covers 100 percent of classes and 88 percent of the lines. The services that are tested are the ones related to creating Puckemons, which is a part of the model and the reason they are tested. All these tests can be found in the "src/test" folder. The github actions workflow can be found at ".github/workflows/tests.yml" and can be viewed at: `https://github.com/Kejovaara/bEsTpRoGrAmMerS_EU/actions`

5.2 Issues

After running PMD on the project, there are merely some violations which are known about and left for clarifying the code (Readability). These include useless parentheses and nested if-statements that could be combined. There are also unused fields, which are "name" for the Player and PuckeTrainer classes. These are left because they are meant to be used in future updates, this is also clarified in the javaDocs.
As of handing in the project no visual or functional bugs were present. Everything that appears in the window is foolproof, there is no discovered way to cause the game to break or work in an unintended way.

5.3 Improvemnts

Following are improvements we wished we had the time to implement to make the code cleaner and simpler.

5.3.1 Defensive Copying

To decrease the risk of altering objects to much should we have used defensive copying on occasions. For example, we battle in combat using a Puckemon changing a lot of its variables only to restore most to their original state. Using a copy instead would allow us to be less careful as once combat is done we can alter the original with the permanent effects and "throw away" the copy not needing to double check the variable changes.

5.3.2 Decreasing object altering

Whole objects are sometimes being altered and used for functions in classes where they really do not belong. We should have instead used single variables directly from the source, instead of asking for the whole object. Another solution would be to again use defensive copying, sending a less valuable object to the class that needs it, keeping the original safe.

5.3.3 Abstract PuckeBag

An improvement that we tried to implement was to abstract Puckemon in the Puckebag classes. The player and opposing trainer each use one particular bag, they are however very similar in some regards. The plan was to make them share code via a parent or interface. However their parties consist of different object lists, Fixed and owned Puckemon. The goal was to make it so that PuckeBag simply contained the Parent class Puckemon allowing the rest of the code not to specify either type, simply type Puckemon. This became problematic quickly as it was more of a problem needing to specify which Puckemon type the list consisted of when using a unique method only accessible by one type. Given more time we would have changed it so that they could share, removing code and adding more abstractions to the code.

5.4 Future Implementation

Due to the time constraint some tasks were simply too big to implement. Given more time we would have tried to implement and improve the following.

5.4.1 World

To implement a traversable world was the first future goal of the project. Like the world in the real Pokémon games it would act like a hub, combat with wild Puckemon and trainers should start there. This was not necessarily intended to be implemented since it is quite big and was left as an extra goal given the time. This is the reason most of the unused code was left, because it would be used in the implementation of the world. One example is the names of the Player and PuckeTrainer class.

5.4.2 Puckemon, Attack and Item Variety

To implement more Puckemons, Attacks and Items would make the game more complete, varied and fun. Because of the extensible design, this would be quite easy, but there was not really any reason to extend the game in that regard. These implementations would use more effects and therefore more methods in the IPuckemon class, which is why there are some methods left, like methods for locking certain stats, meant to make them unchangeable by opposing attacks.

Given some free time it would be fun and simple to implement more, it would however not improve on the code.

5.4.3 Catching Puckemon

The original idea was to let the player catch wild puckemon during battle and add them to their party. In the end there was simply no time to implement and test it. There are still methods and variables in PlayerPuckeBag ready to be used should it be implemted.

5.4.4 Saving Game

A function to save the game would be a good implementation, though at this stage of the game it might be a bit redundant. Together with the implementation of a world, saving the game would serve as a nice addition to the application.

6 References

References

[1] LibGdx game library (2021). Available at: `https://libgdx.badlogicgames.com/ci/nightlies/docs/api/`

[2] Apache POI (2021). Available at: `https://mvnrepository.com/artifact/org.apache.poi/poi`

[3] Maven (2021). Available at: `https://maven.apache.org/`

7 Appendix

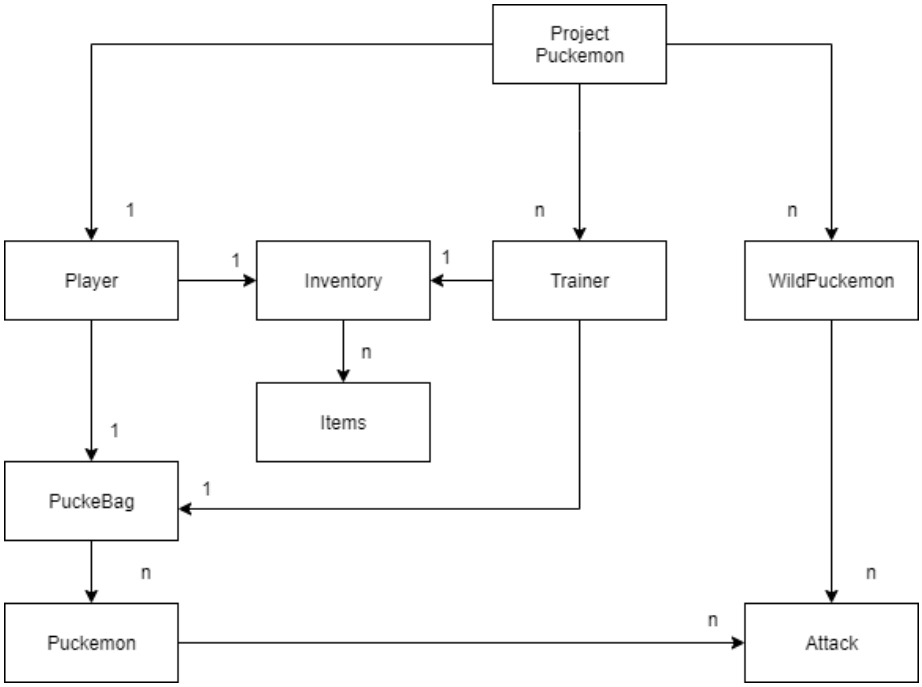

Figure 1: Domain model

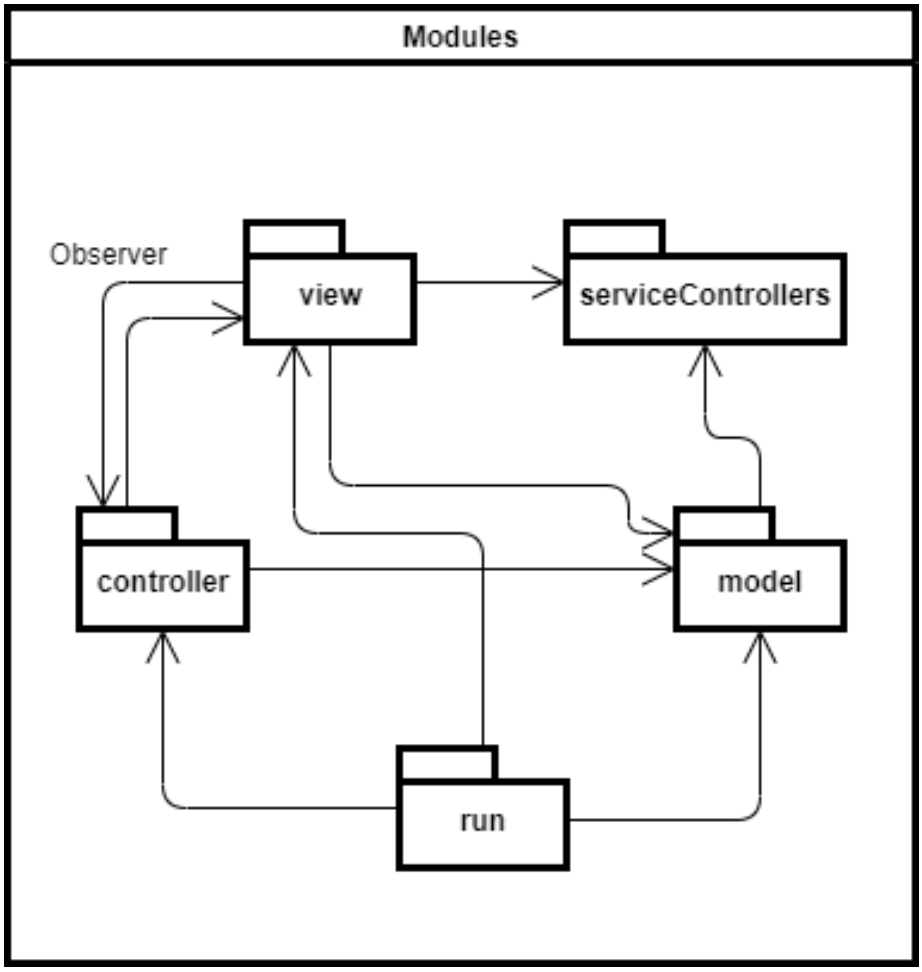

Figure 2: Top level modules

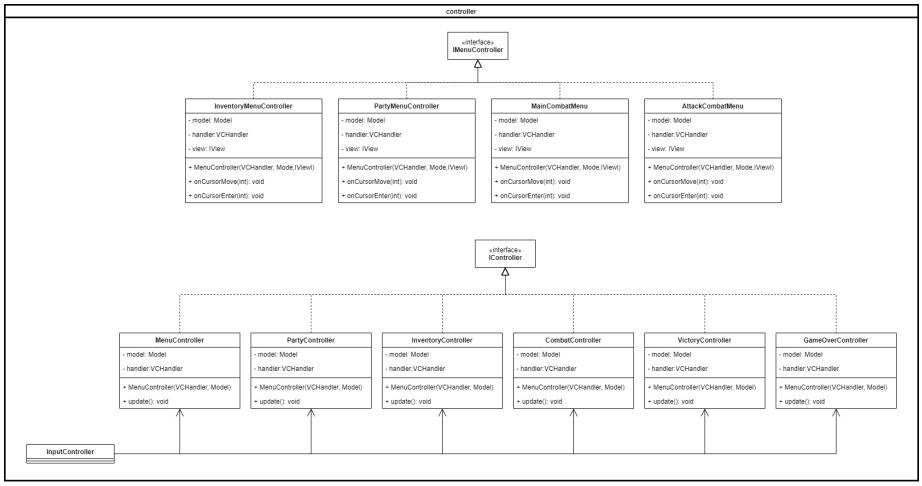

Figure 3: Controller package

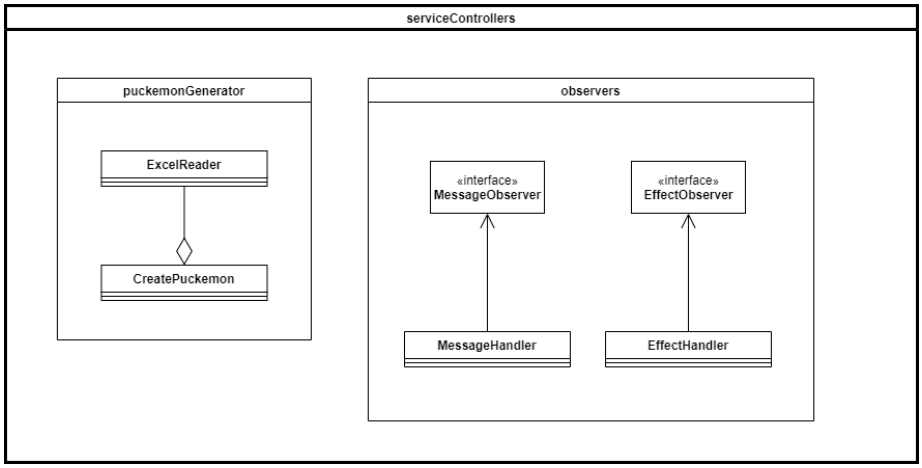

Figure 4: ServiceControllers package

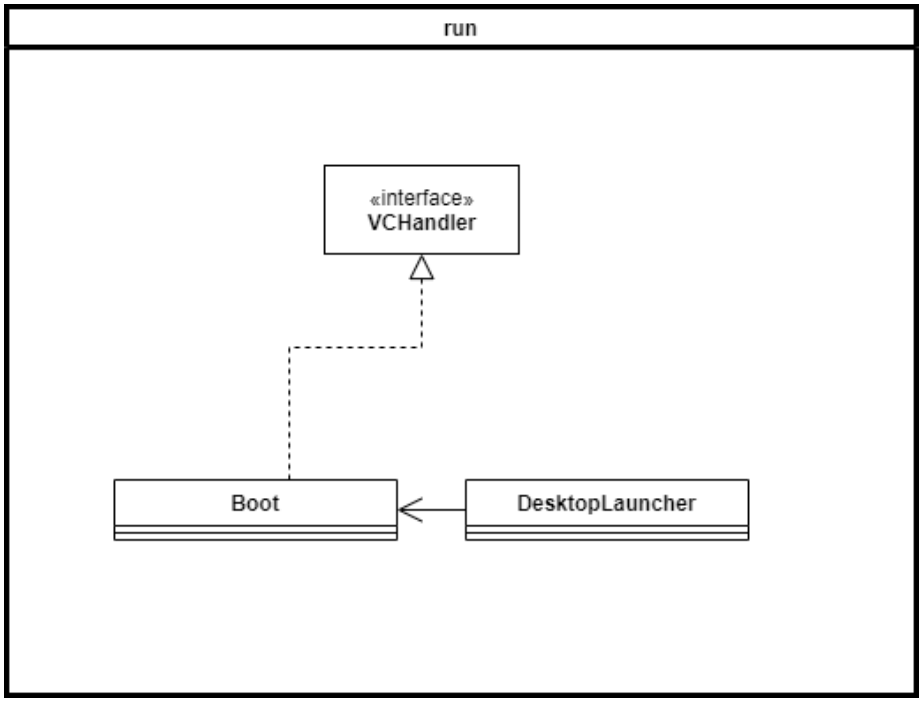

Figure 5: Run package

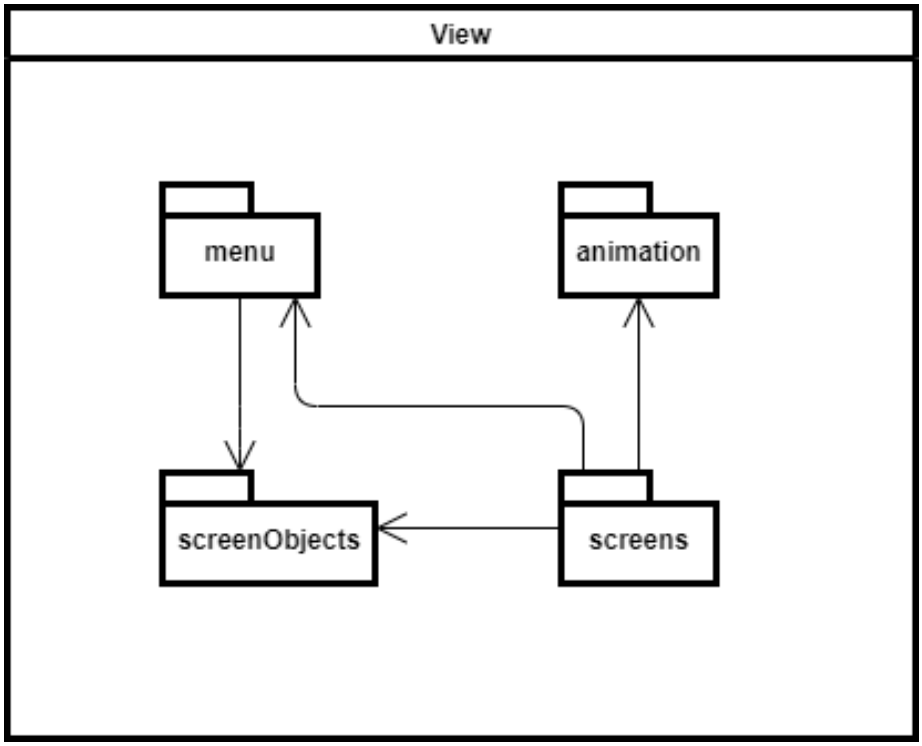

Figure 6: View package

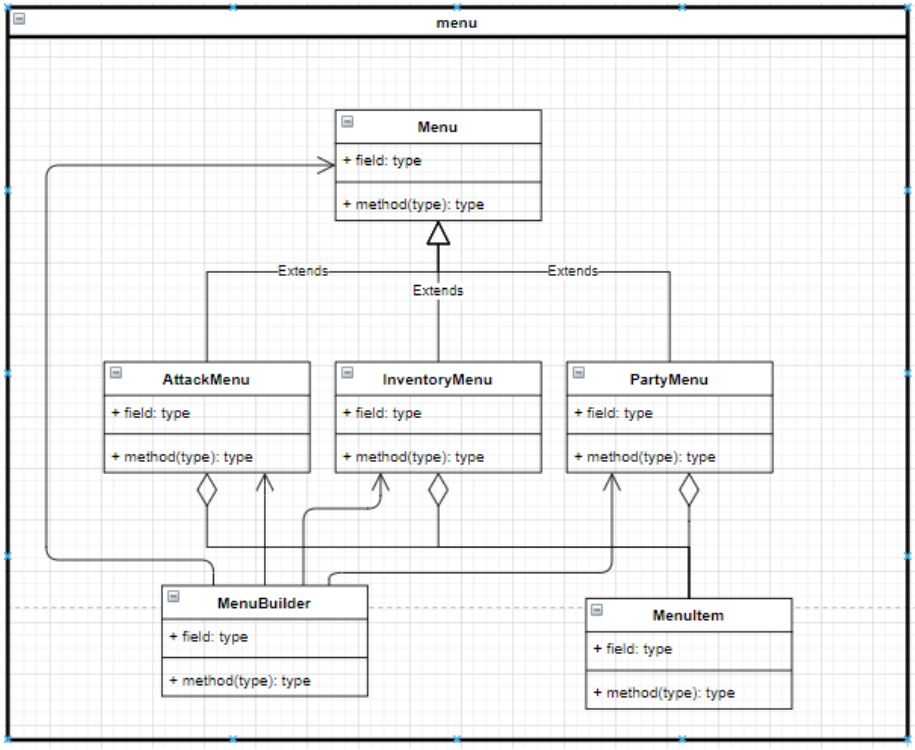

Figure 7: Menu packagel

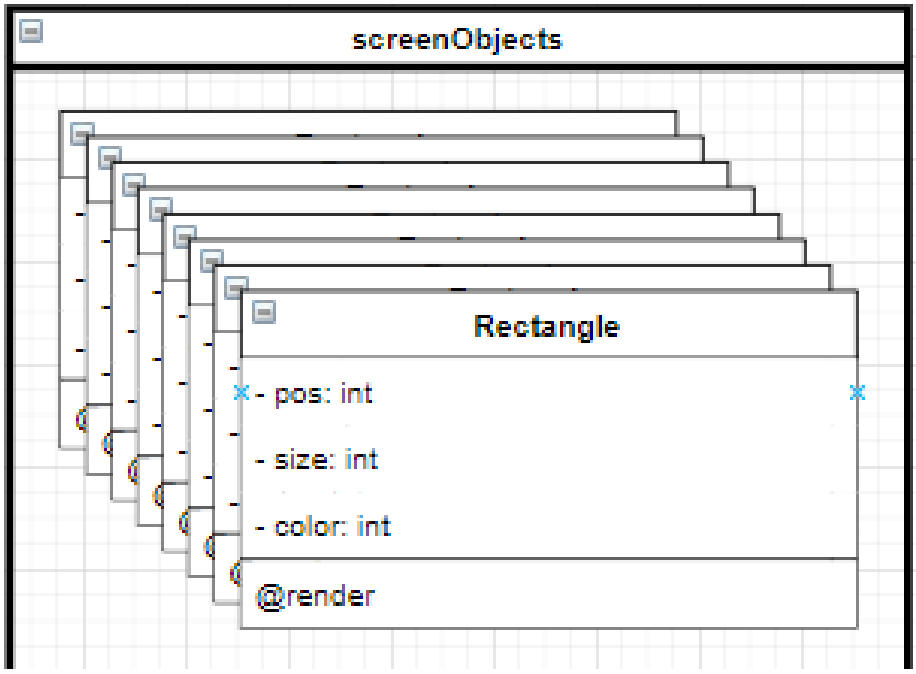

Figure 8: ScreenObjects packagel

Figure 9: Screen package

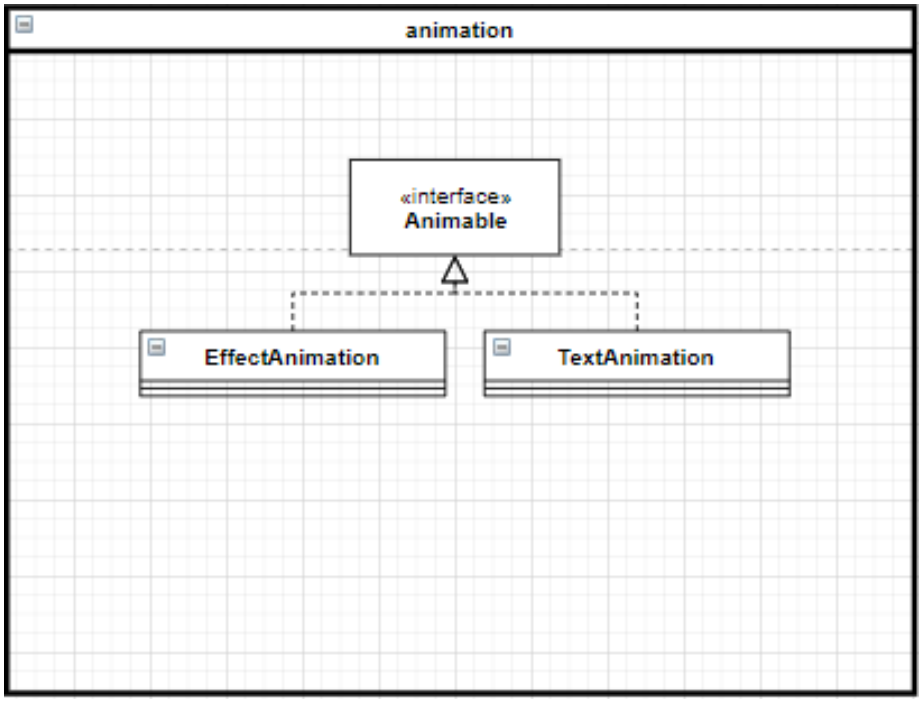

Figure 10: Animation package

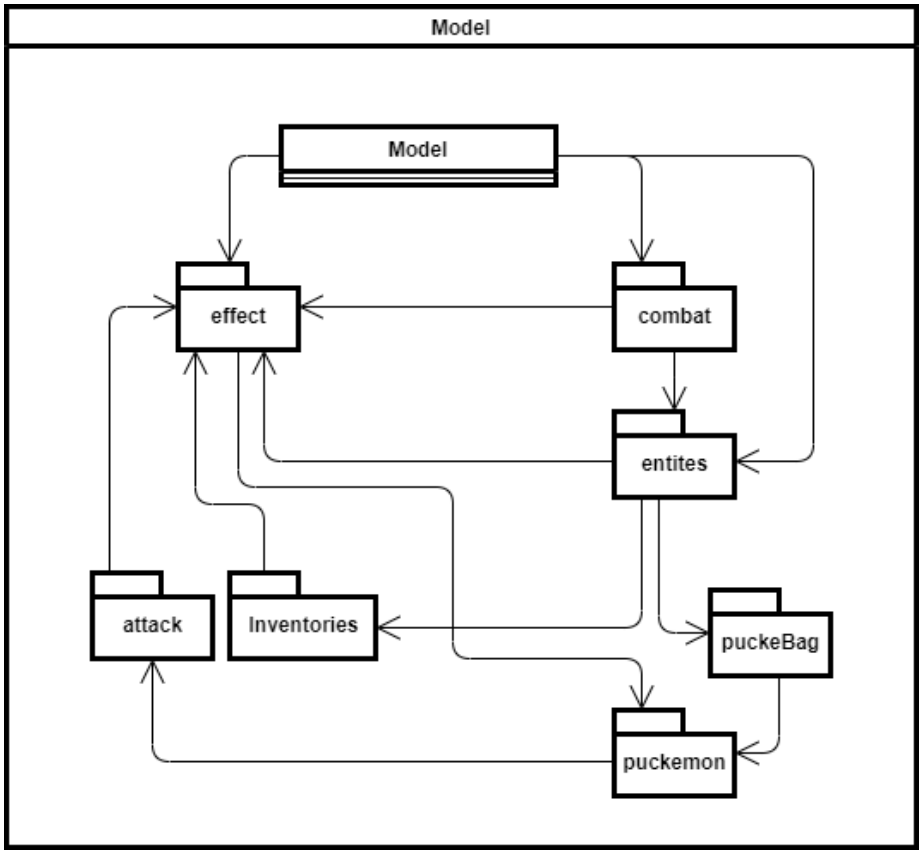

Figure 11: Model package

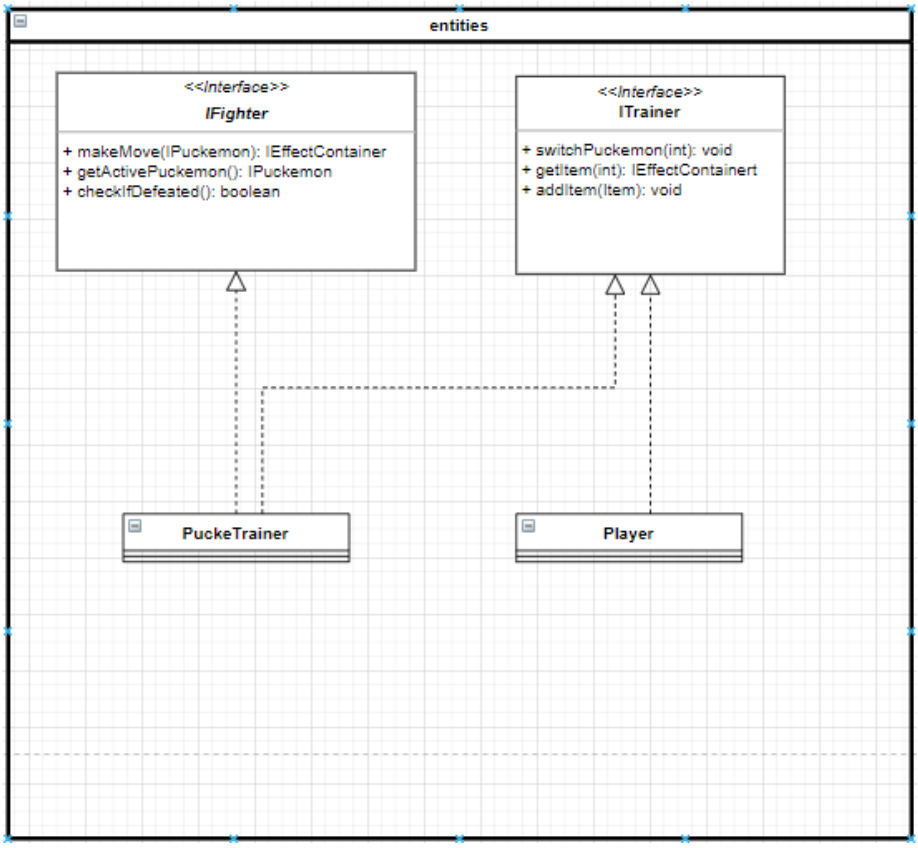

Figure 12: Entities package

Figure 13: Puckebags package

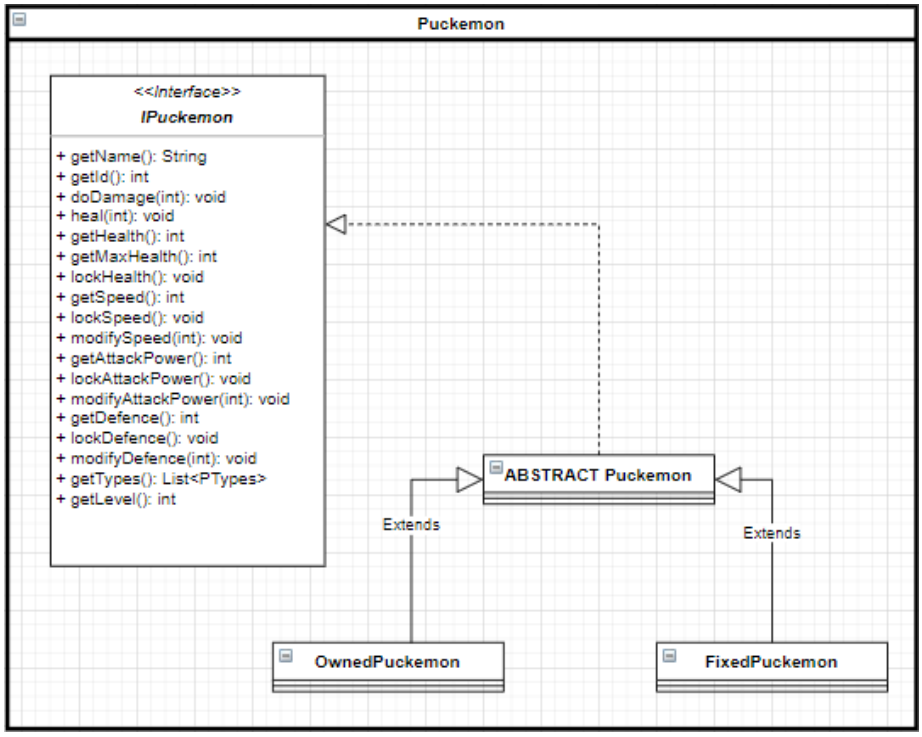

Figure 14: Puckemons package

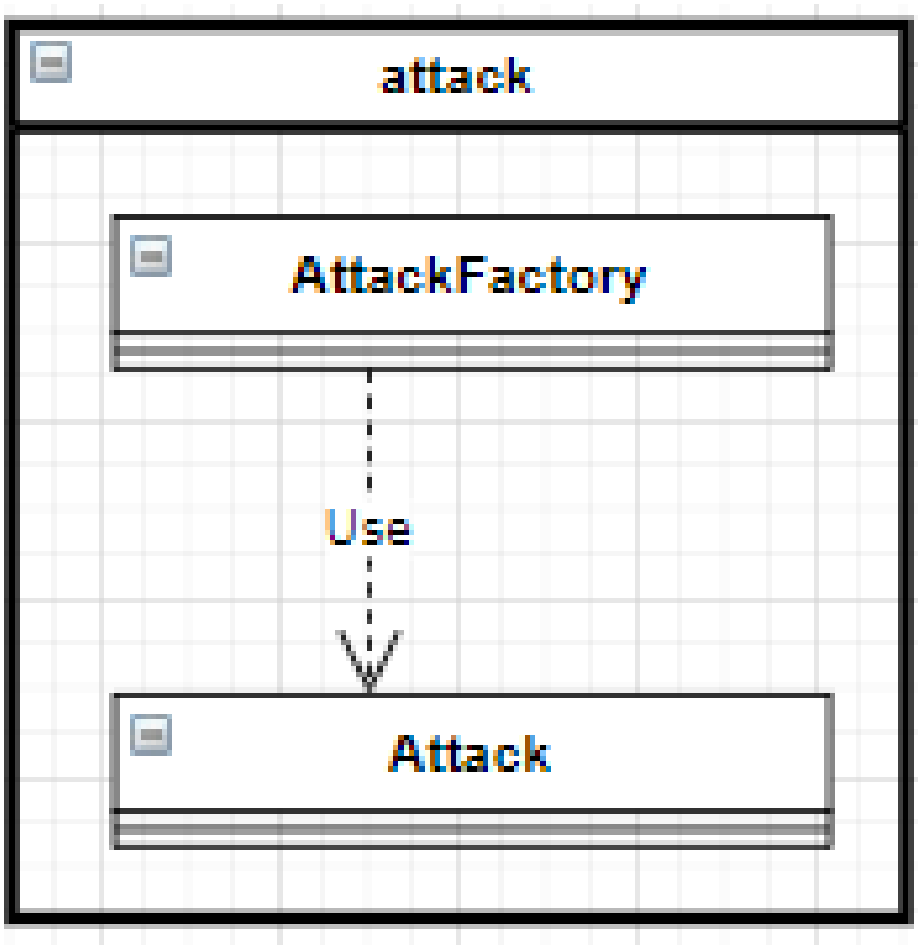

Figure 15: Attack package

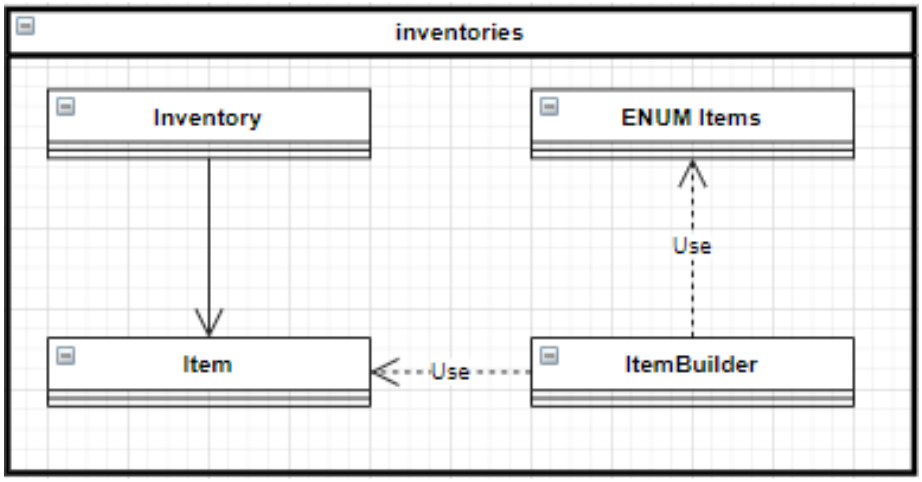

Figure 16: Inventories package

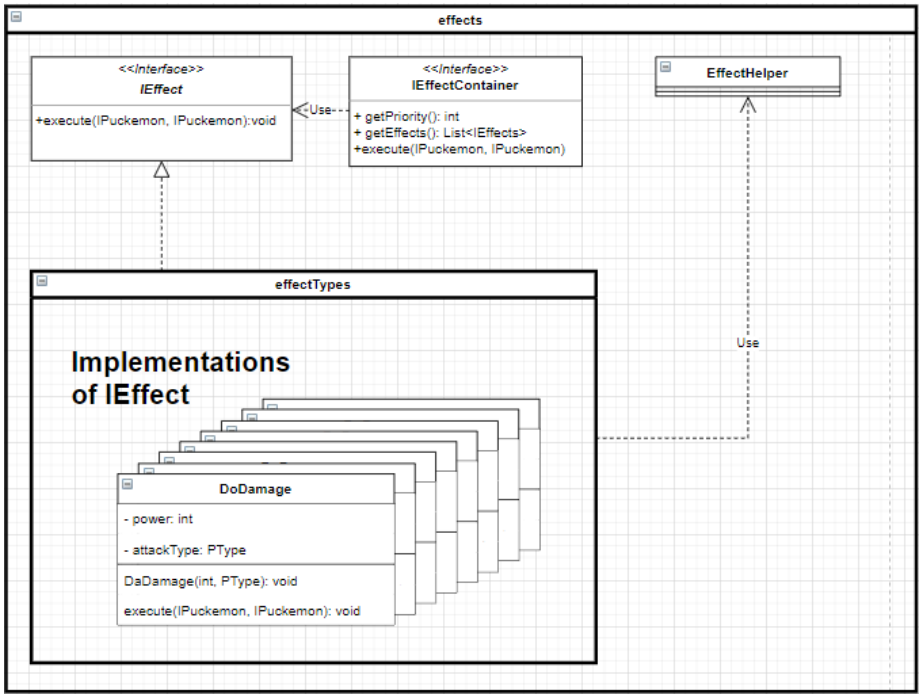

Figure 17: Effects package