TDA367-Report

Erik Jergéus, Eric Carlsson, Jacob Pedersen, Theodor Angergård, Vidar Magnusson

October 22, 2018

Grupp 2 - TEVEJ

Contents

1 Introduction

The project is a simulation/strategy game where the user can build buildings, gather resources and then use the resources to progress in the game. You play the game from a god-perspective with the graphics being of a top-down 2d nature.

1.1 Definition, acronyms, and abbreviations

1.1.1 World

The world refers to the grid of tiles that makes up the game.

1.1.2 Inventory

The inventory is the user's collection of resources that can be used to construct new buildings or feed the population.

1.1.3 Building

A building is something that the user can place into the world. It may either be a resource gathering building, e.g. a mining building that gathers stone. Or it might be a house building, a place for your population to live.

1.1.4 Resource

A resource is something you can use to build buildings or feed your citizens. E.g. Wood, Food, Stone and the Population itself.

1.1.5 Natural resource

Natural resources are objects in the world that can be mined by placing buildings near them. When they are mined they add resources to the user's inventory.

1.1.6 Tile

The world has a tile-based structure, i.e. a grid. Each tile may hold a building or a natural resource but more often then not, they are empty.

1.1.7 Engine

A central hub for the model's information imported from a framework called "Ashley" created by Badlogic.

1.1.8 Signal

A small piece of information containing an entity and a signal type. This is used in some parts of the model to broadcast information to other relevant parts of the model.

1.1.9 GUI/UI

UI stands for User Interface.
GUI stands for Graphical User Interface.

2 Requirements

2.1 User stories

User story
Story ID: 0
Story Name: Start application
Description
As a user I need to be able to start the application.
Confirmation
- Can I run a java file?
- Can I open the file?

User story
Story ID: 1
Story Name: See the world
Description
As a user I want to be able to see the world.
Confirmation
- Does a world exist?
- Does tiles exist?
- Does the world have a large amount of tiles in a grid?
- Can I see the tiles?

User story
Story ID: 2
Story Name: See resources
Description
As a user I want to see resources in the world.
Confirmation
- Does resources exist?
- Can tiles have resources on them?
- Can the resource be seen as they are supposed to be?

User story
Story ID: 3
Story Name: See buildings
Description
As a user I want to see buildings.
Confirmation
- Does buildings exist?

- Can tiles have buildings on them?
- Can the buildings be seen as they are supposed to be?

User story
Story ID: 4
Story Name: Navigate the world
Description
As a user I want to be able to navigate the world.
Confirmation
- Can inputs be registered?
- Can i move the camera by dragging my mouse in the world?

User story
Story ID: 5
Story Name: Home building
Description
As a user I want to there to be a Home building.
Confirmation
- Can I have a home building?
- Can I have population in my home?

User story
Story ID: 6
Story Name: Start with home
Description
As a user I want to start with a home.
Confirmation
- Can I start with a home on a tile?

User story
Story ID: 7
Story Name: Inventory
Description
As a user I want to have an Inventory.
Confirmation
- Can I have an inventory?
- Can I see my inventory?
- Can my buildings cost and generate resources in my inventory?

User story
Story ID: 8
Story Name: Deplete Resources
Description
As a user I want some resources to be able to deplete.
Confirmation
- Does finite resources exist?

- Can I not gather any more resources when it is empty? - If I deplete a resource, is it deleted?

User story
Story ID: 9
Story Name: Place buildings
Description
As a user I want to place buildings in the world.
Confirmation
- Is a keybinding bound for each building?
- Can I place a building where my mouse is?

User story
Story ID: 10
Story Name: Cost of Buildings
Description
As a user I want buildings to cost resources.
Confirmation
- Can I only build a building if I have enough resources?
- If i build a building, are the proper amount of resources removed from my inventory?
- Can I see that buildings cost resources?

User story
Story ID: 11
Story Name: See buildings to build
Description
As a user I want to see what buildings I can build.
Confirmation
- Can I see the available buildings?
- Can I see the buildings I can not afford?

User story
Story ID: 12
Story Name: Pliancy when placing buildings
Description
As a user I want to know if i can place buildings.
Confirmation
- Can I see buildings next to the mouse when placing?
- Can I see that a spot is unavailable for placing?

User story
Story ID: 14
Story Name: Zooming
Description
As a user I want to be able to zoom in and out.

Confirmation
- Can I zoom in?
- Can I zoom out?
- Can I still do the same things when zoomed in/out?

User story
Story ID: 15
Story Name: Growing trees
Description
As a user I want trees to be able to grow by themselves.
Confirmation
- Can trees grow by themselves?

User story
Story ID: 16
Story Name: Upgrade buildings
Description
As a user I want to be able to upgrade buildings.
Confirmation
- Can I see a GUI for upgrading buildings?
- Can I see that a building is upgraded?
- If I upgrade a building, does it gain functionality?

2.2 User interface

2.2.1 Main user interface

Figure 1: The initial sketch of the UI.

The initial user interface sketch consisted of three basic elements: One to show the inventory, another to show information about buildings and one to construct buildings.

- The "Inventory" UI exist to show the user what they have in their inventory. That way they will now if they have enough resources to keep their population alive or if they have enough to build a specific building. It was designed to be simple but informative by using both images and text that updates in real-time so that the user always knows what they have in their inventory. The numbers telling the user how much of each resource that remains have suffixes for thousands (K) and millions (M) to avoid filling up the screen with long strings.
- The "Building Information" UI shows the user what a building is named and how much it costs. It also shows an image of the building to help the user associate it with an actual building on the map. It pops up whenever the user either selects a building from the "Construct Building" UI or clicks on an existing building on the map.

- The "Construct Building" UI shows the user all of the different buildings that are available for construction. If the user clicks on one of the building-icons the icon will then follow the cursor until the user clicks on an empty tile on the map or deselects the building by pressing the icon again. If the user clicks on an empty tile the selected building will be built on the tile that was pressed.

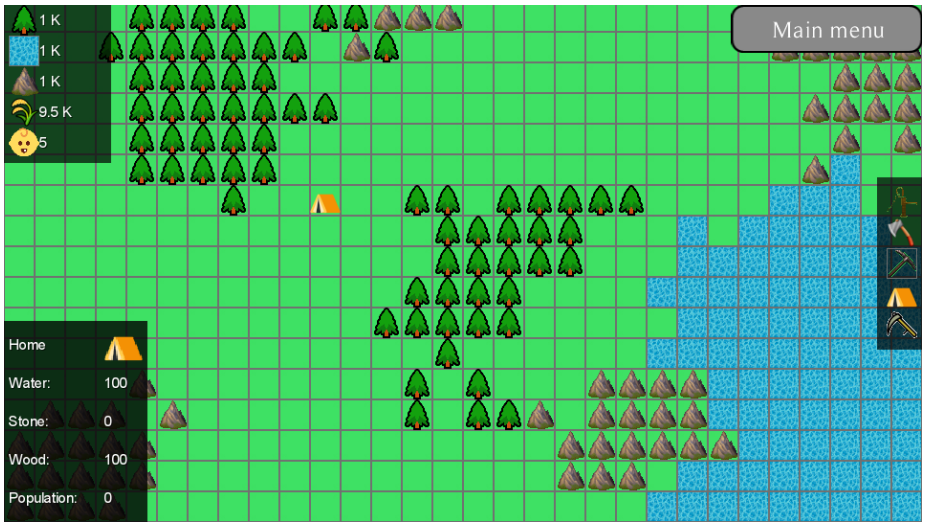


Figure 2: This was the final result of the main UI.

During the development process the UI changed in several ways. The UI element representing the inventory was moved from the top to the top-left corner. This change was mostly made to make the interface more static even when the text gets wider (when the inventory grows). Another difference between the sketch and the result is the size of the "Construct Building" UI-element on the left side. It is only one column instead of two and it is not as tall as in the sketch. This is a side-effect of the fact that the game does not include as many buildings to choose from as first intended. The "Main menu"-button in the top-right corner of the screen was also added to the UI. It is a simple button that takes the user to the main menu of the game.

2.2.2 Menu user interface

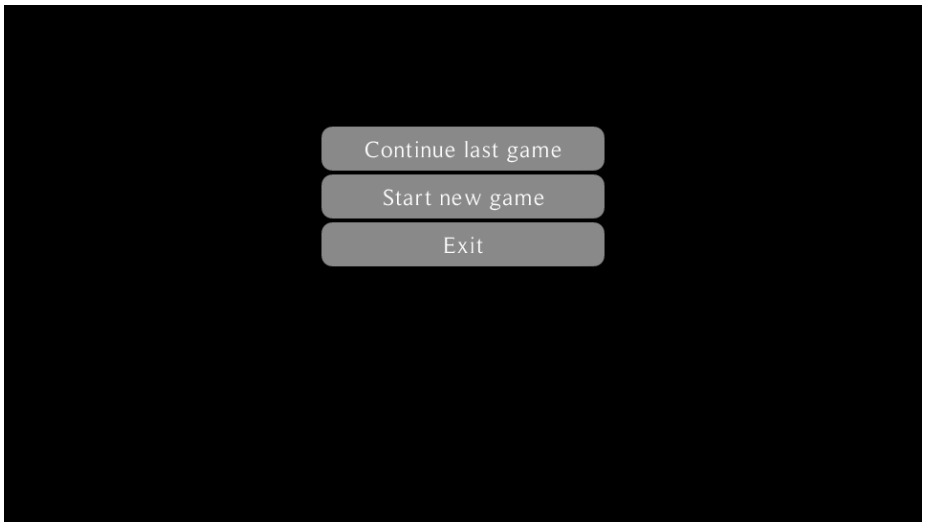


Figure 3: The "Menu" interface

The "Menu" interface was added to the game to let the user choose whether to:

- **"Continue last game"** - This would load the save-file from the last time the user played the game.
- **"Start new game"** - This would begin a new game and delete the previous save-file.
- **"Exit"** - This would save the user's progress and quit the application.

This interface uses simple buttons to display the different alternatives presented to the user.

3 Domain model

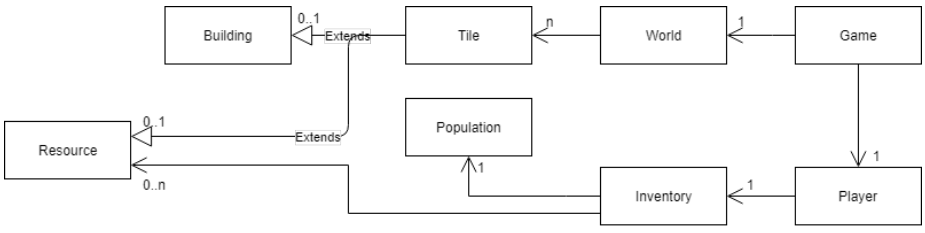


Figure 4: The domain model.

3.1 Class responsibilities

Game: Class that initializes and runs the entire project.

World: A logical representation of all objects in the world that structures them for convenient use.

Tile: Represents a location on the map.

Building: One of the things you can place on tiles to build your world.

Resource: Represents the cost for buildings and the amount you can extract from natural resources such as Trees and Lakes.

Inventory: Represents the current amount of resources the user has at his disposal.

4 System architecture

4.0.1 MVC

In the project MVC is implemented by having the controller initialize the model and the view. After initialization the controller contains the main game loop and updates the model and view accordingly. The controller also listens to the view for user input via observer patterns, after receiving a user input the controller updates itself, the model and the view accordingly. The view listens on the model and updates the graphics/user interface accordingly.

4.0.2 Entity Component System

The model of the game is built using Entity Component System (ECS), which favors composition over inheritance. Here follows a description on ECS and how it's used in the project.

- **Component** - Components are thin classes that generally only contain data and little to no logic. Examples of components in the project are: - PositionComponent, which has two coordinates (x and y). - SizeComponent, which has a width and a height.
- **Entity** - Entities are classes that holds components and represents the objects in the game. An entity is defined by the components it holds, for example the Home building has the components PositionComponent, SizeComponent, BuildingComponent and a HomeComponent.
- **System** - The systems manipulates the components data and constitutes (almost) the entirety of the gamelogic. The systems are either updated with the game loop for example, the NaturalResourceGatheringSystem,

that collects resources for the gatheringbuildingsw from nearby natural resources. Another way the systems are updated are through signals, that work like global listeners where interested systems listens to specific signals and are then updated accordingly. An example of a system updated through a signal is the BuildBuildingSystem which receives a signal to build a building at a specific location and then does just this.

4.1 Subsystem decomposition

This project only has one system component, the game.

4.2 The Design Model

The design model of the project has been split up into many different parts (mainly based on the packages in the project). This is to make them easier to understand and maintain as it might otherwise become relatively large as the project is built up by around a hundred java files. Starting with the highest level of the project.

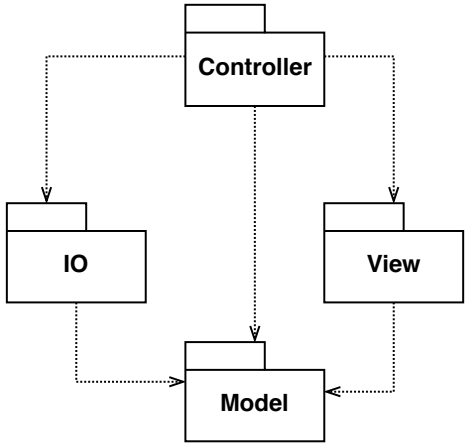


Figure 5: The main package, following a relatively standard MVC format.

4.2.1 Controller

The controller packages top level package looks like this.

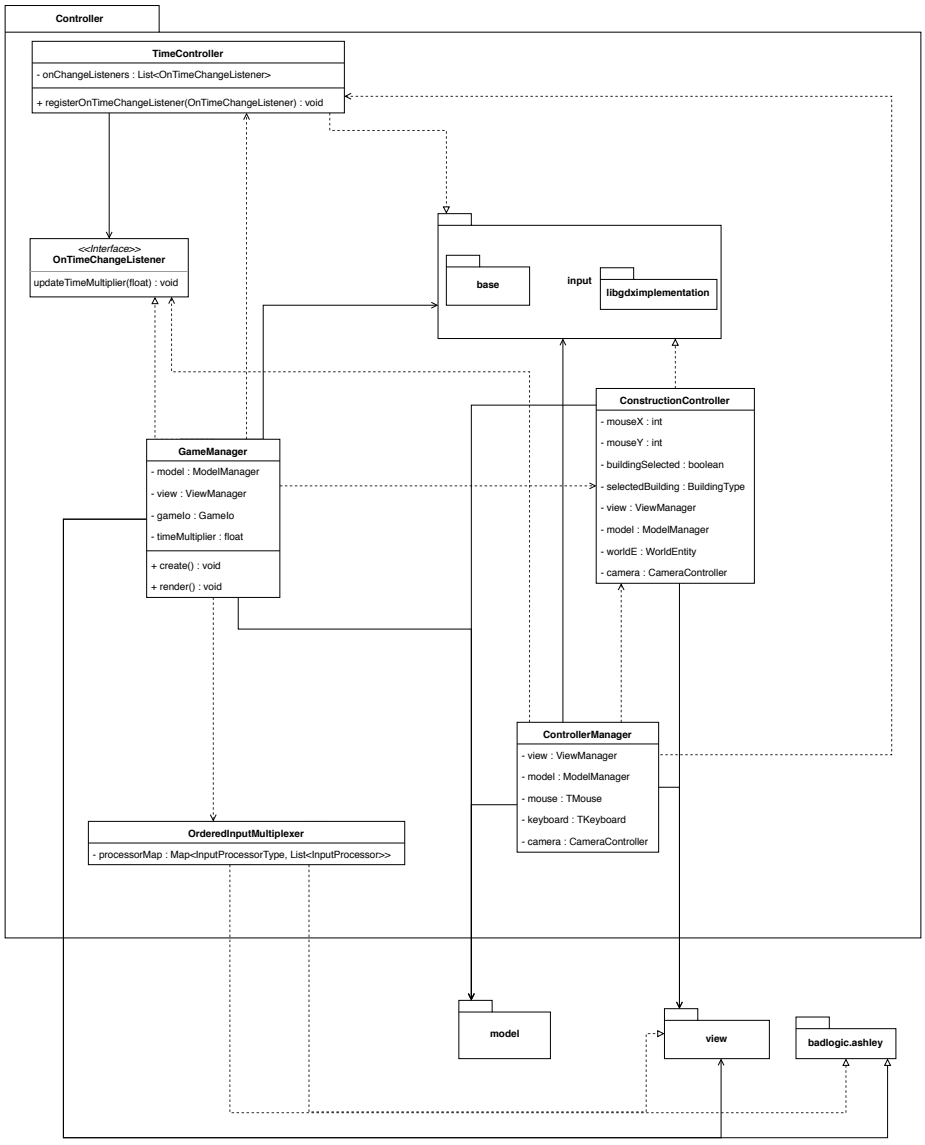


Figure 6: The main controller package.

The main class of the controller is the GameManager which initializes the ControllerManager, ModelManager and ViewManager and contains the main game loop which it uses to update the managers. The ControllerManager is

then responsible for keeping track of the rest of the controller package and it's sub-packages.

Within the Controller package there is also an input package that only contain two other packages, base and libgdximplementation.

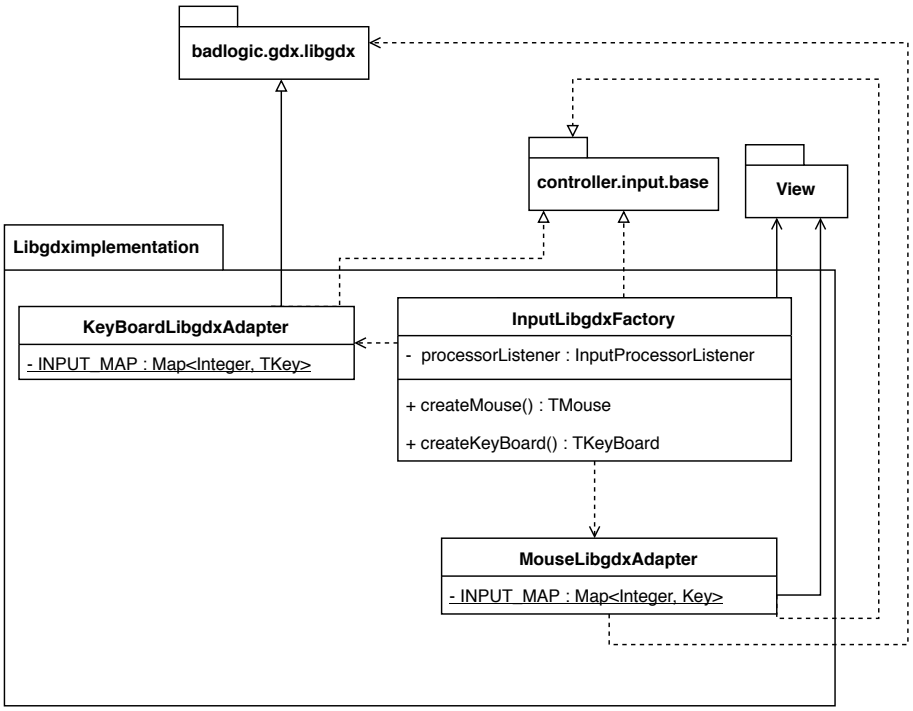


Figure 7: The controller.input.libgdximplementation package

The libgdximplementation package does mostly that, implements libgdx functionality in the game. This package is supposed to work as a sort of adapter between the controller and libgdx and provide a clear interface (seperate from libgdx) for other classes that needs to use keyboard or mouse input. This interface is defined by interfaces in the input.base package (seen below).

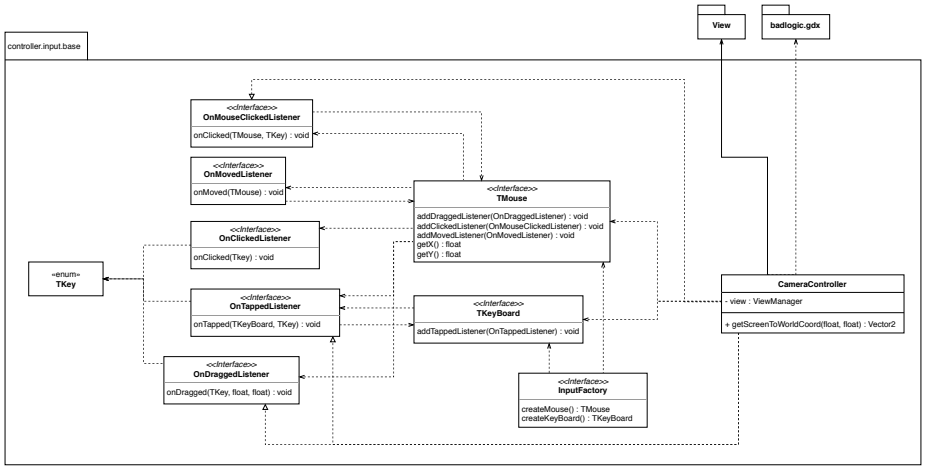


Figure 8: The controller.input.base package

The input.base package mostly contains interfaces that provide the previously mentioned seperation from libgdx, for this purpose it also has the TKey enum. The package also contains the CameraController which is responsible for moving and scaling the "camera" (the players perspective of the game world).

4.2.2 Model

The model is the actual data and game logic.

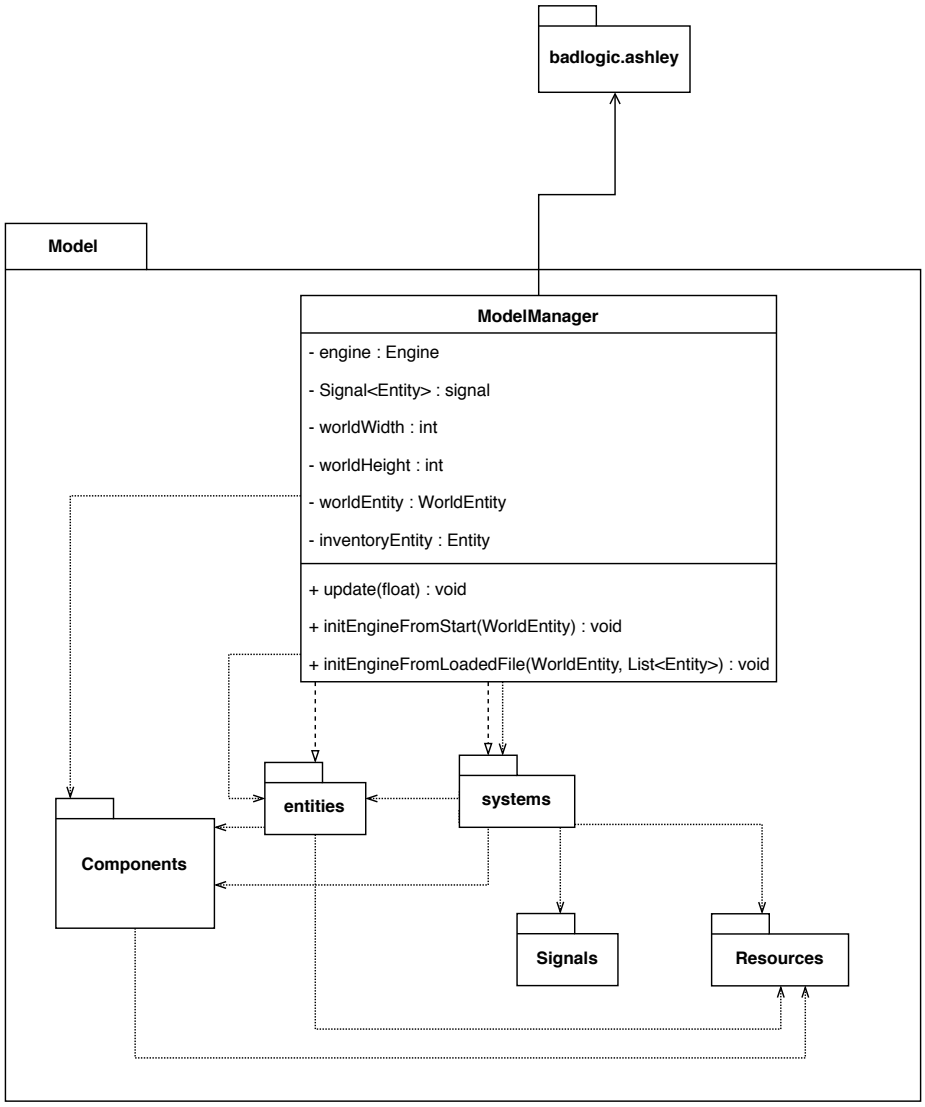


Figure 9: The main model package.

The main class of the model is the ModelManager which has an "engine" that comes from the Ashley framework. Through the Engine the ModelManager keeps track of all the entities and systems in the game. The ModelManager is also (as can be seen above) the only class in the base model folder.

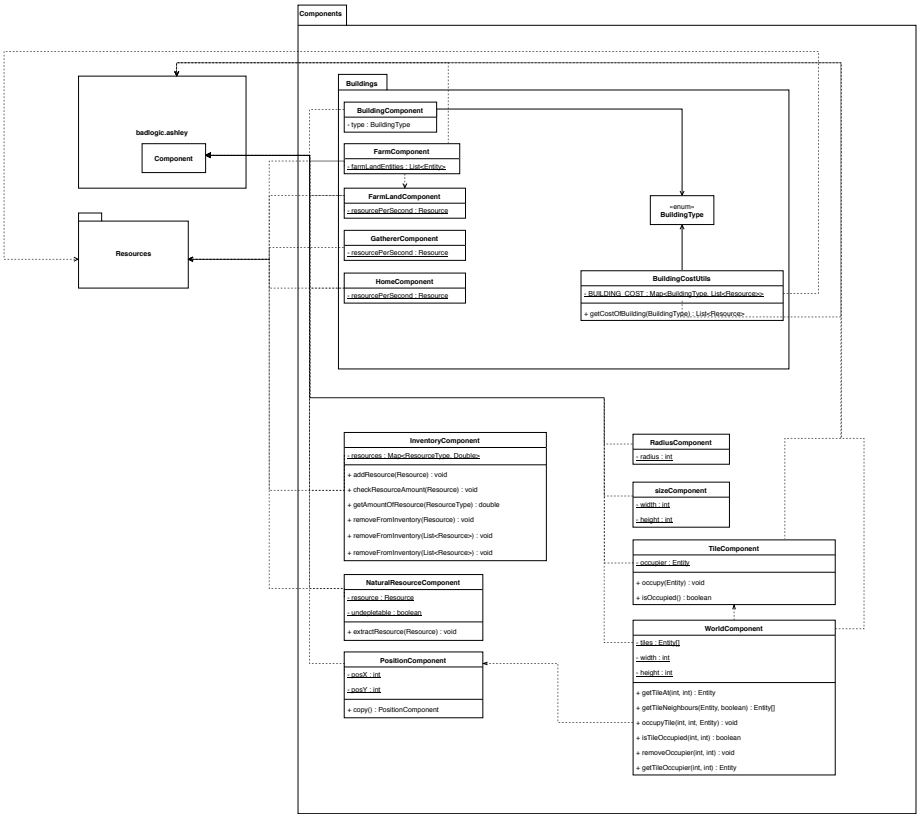


Figure 10: The model.components package.

The component package contains all (except one) of the components in the game, it also has an inner package (model.components.buildings) to seperate the building related components from the rest. All the components inherits from the Ashley frameworks Component class.

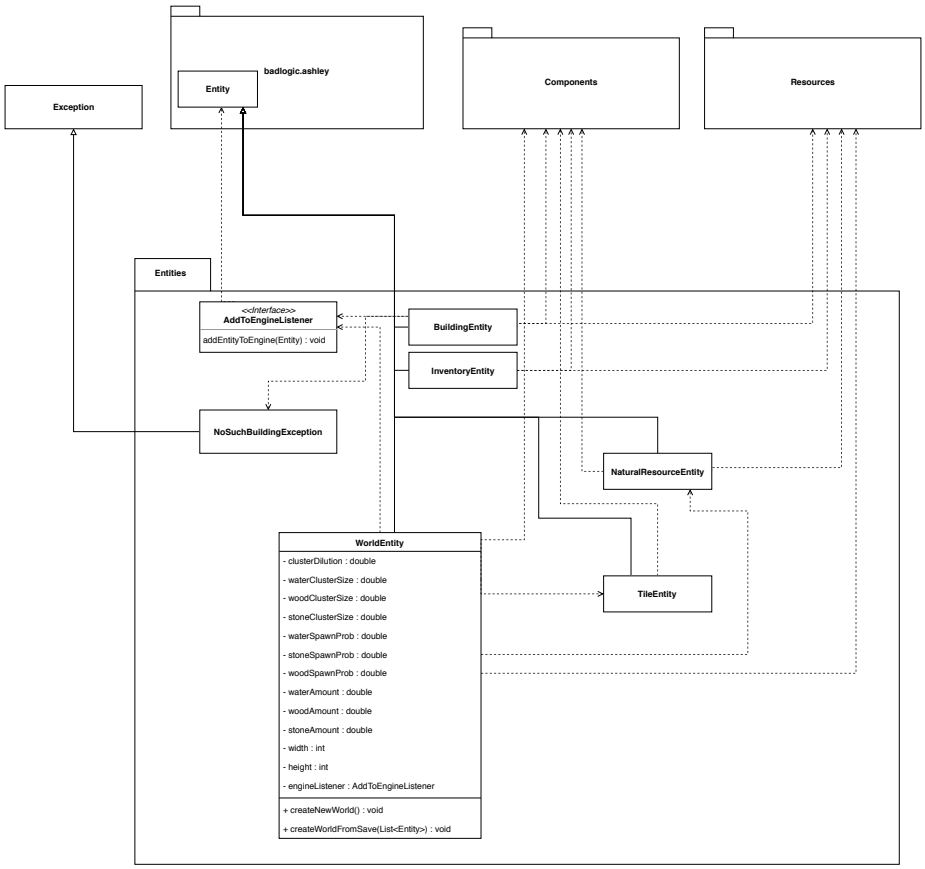


Figure 11: The model.entities package.

The entities package contains entity class that inherits from the Ashley frameworks entity class but acts as factories for the game. In the entities constructor the appropriate components are added to that entity, for example the NaturalResourceEntity simply adds a NaturalResourceComponent, a PositionComponent and a SizeComponent to itself.

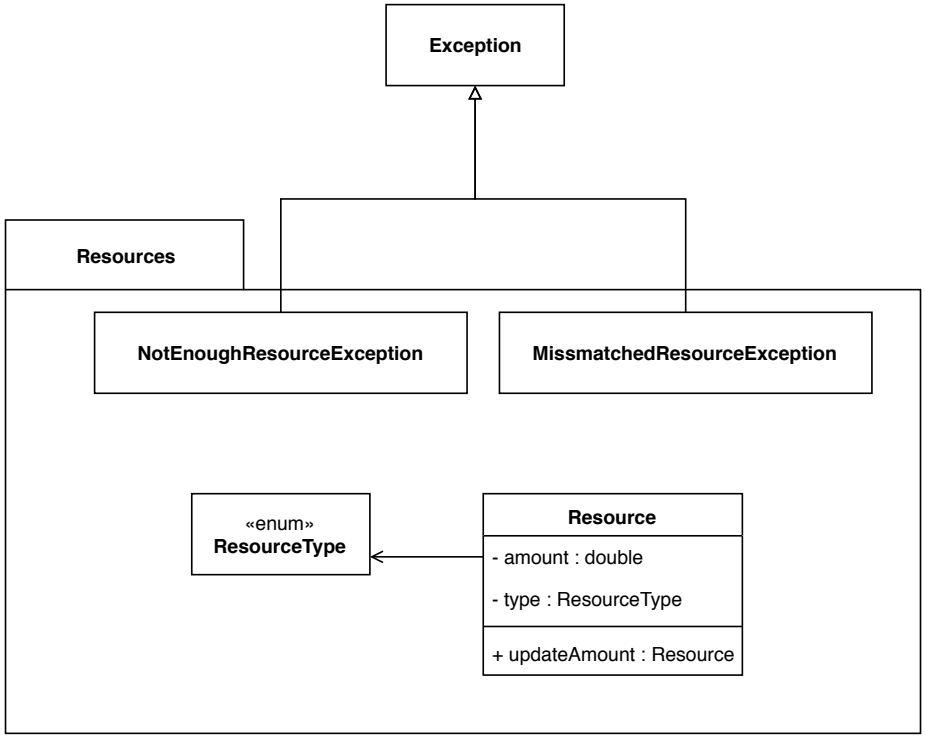


Figure 12: The model.resources package.

The model.resources package simply contain the immutable Resource class as well as related classes.

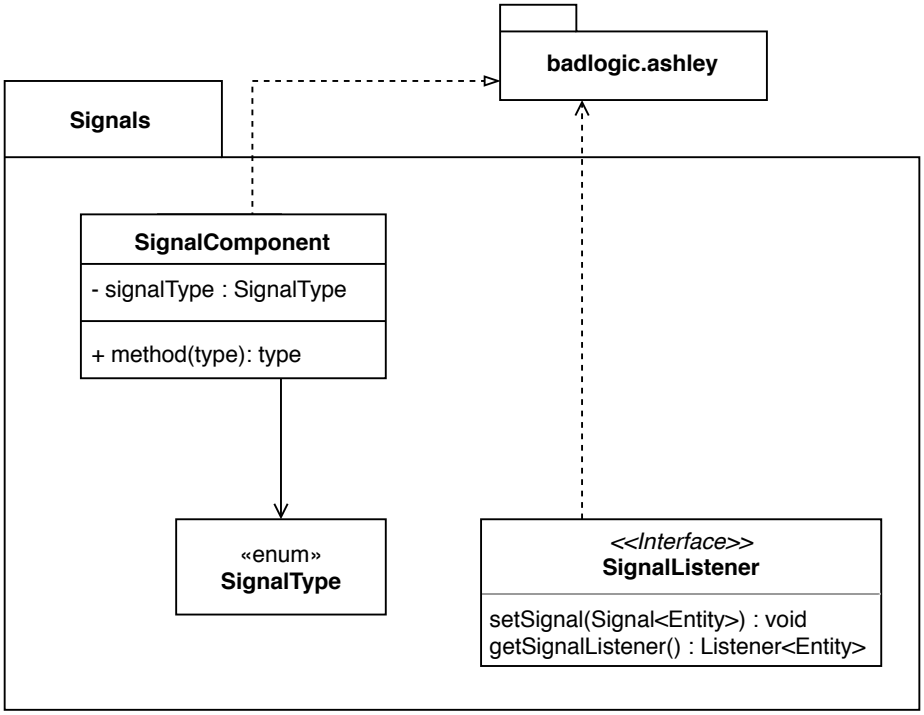


Figure 13: The model.signals package.

The model.signals package contains the SignalComponent (The only component outside the components package) which is used on entities that are passed on when a signal is dispatched.

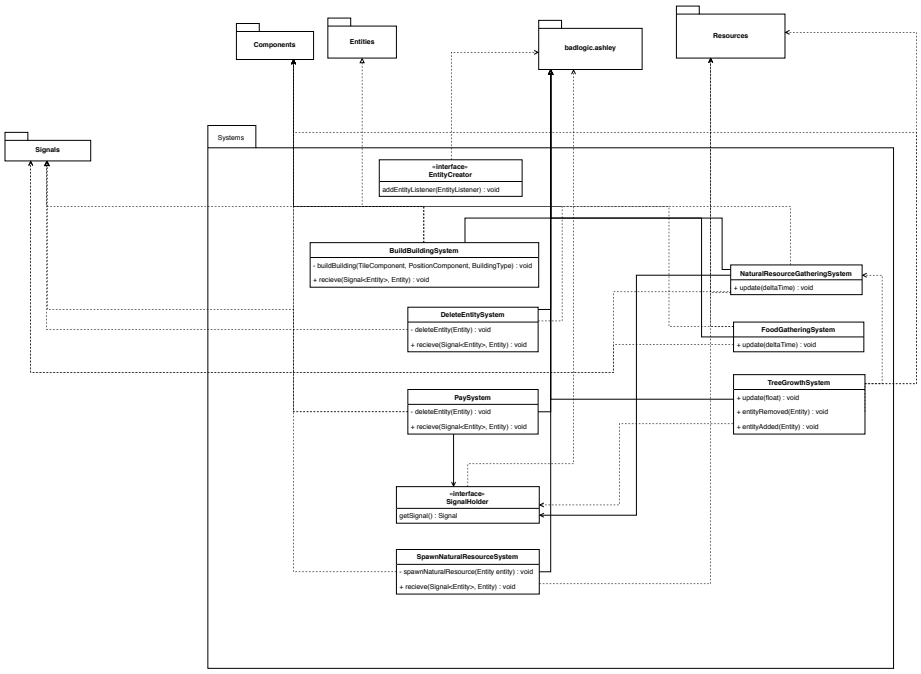
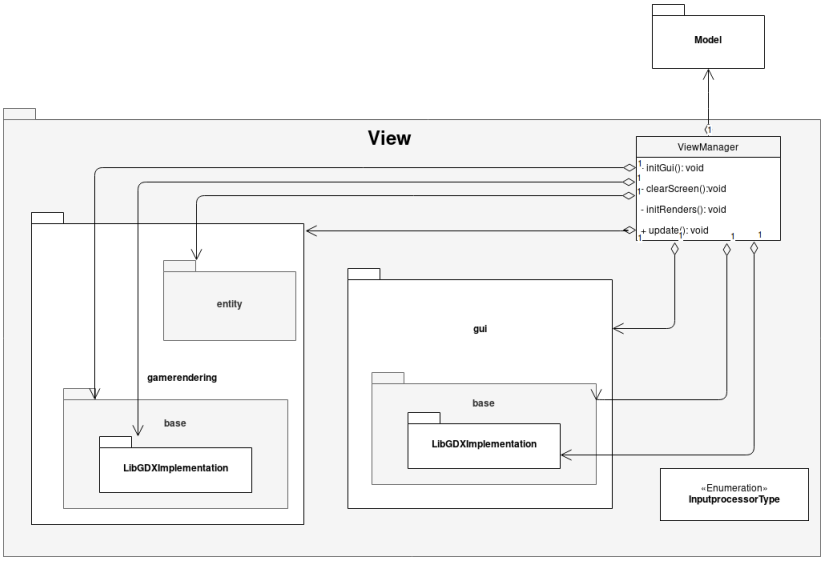


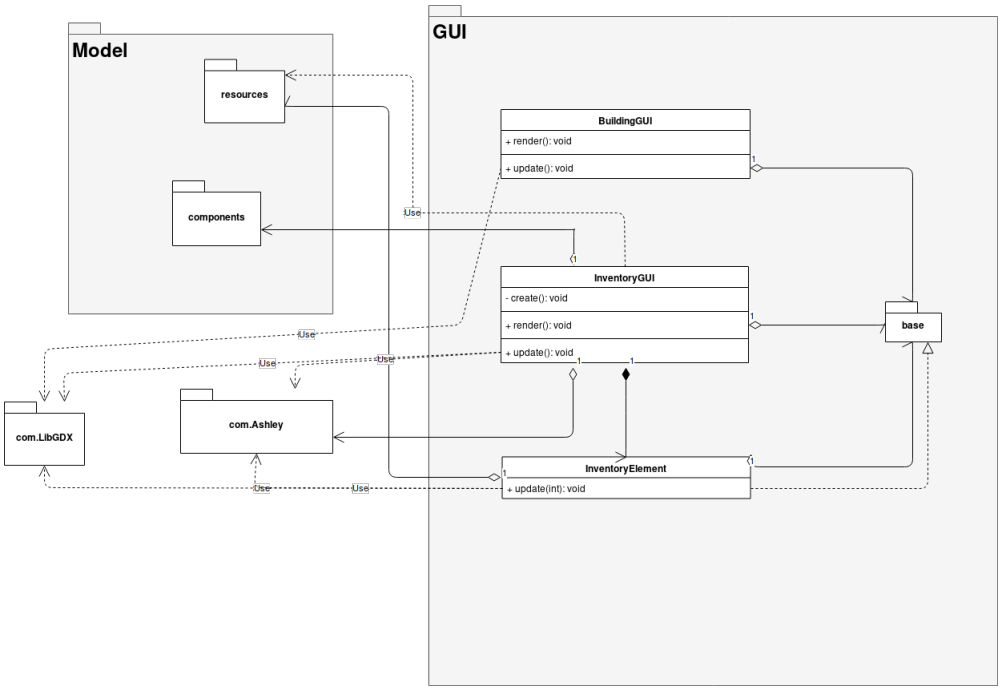
Figure 14: The model.systems package.

The systems package contains all the systems in the game, which in turn manages/manipulates the components(/entities) and therefor has a relatively large amount of dependencies on the rest of the model.

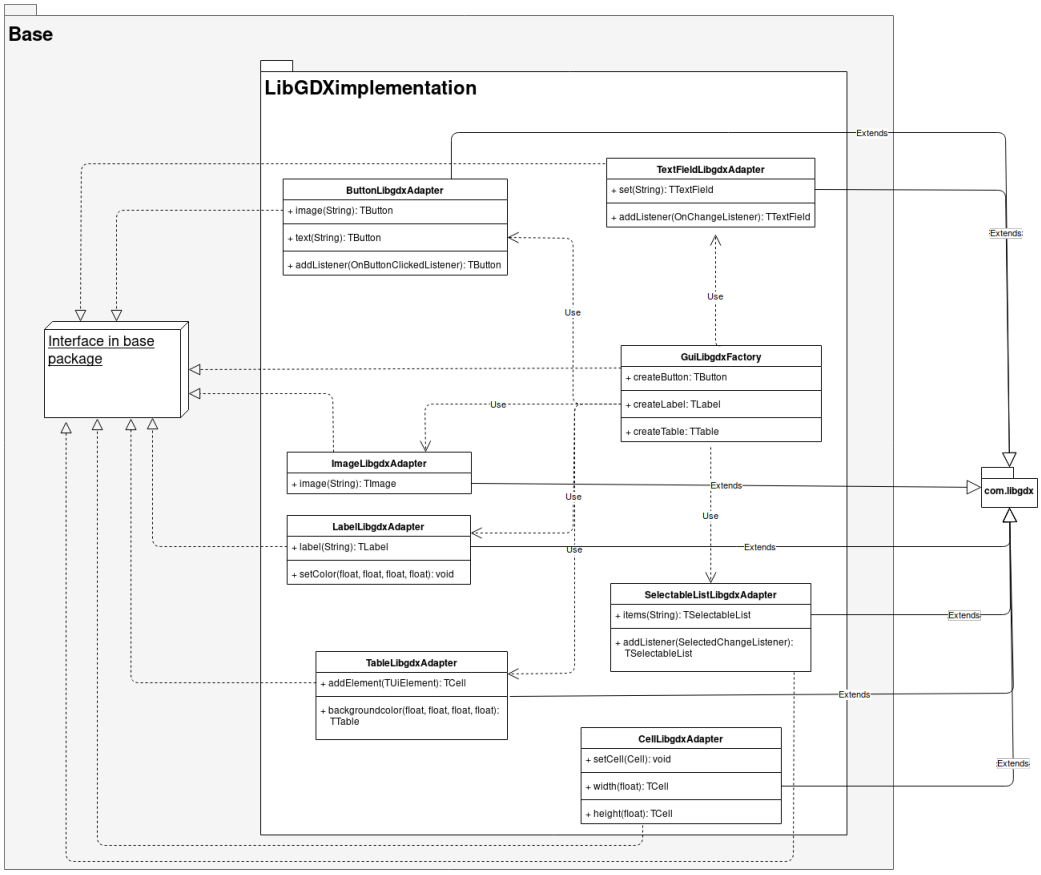
4.2.3 View



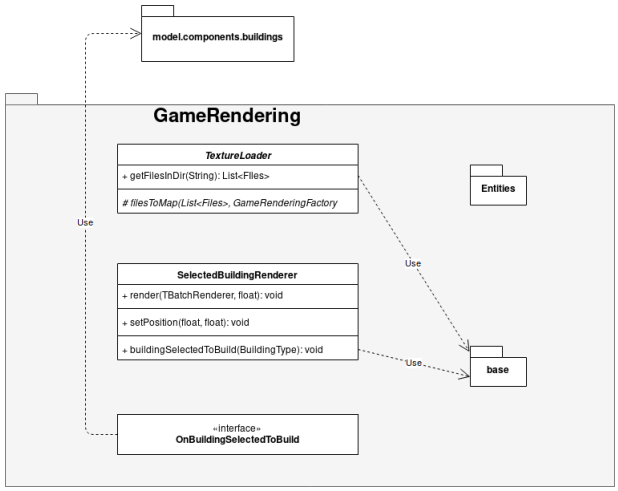
The view has the responsibility to represent the model and by modifying it via input events. You can split the view into two parts, gui and gamerendering.



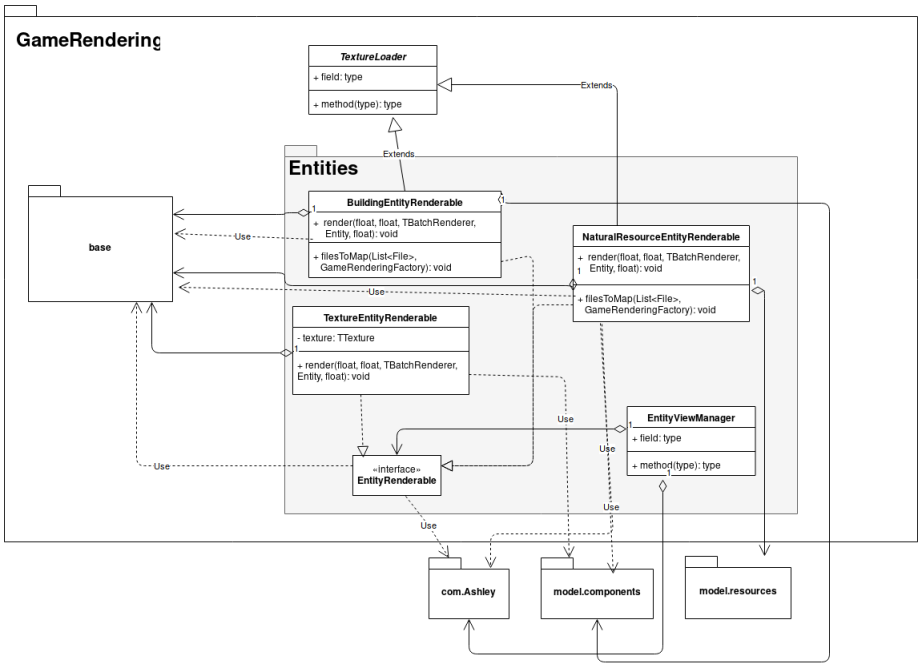
The **gui** part renders different kinds of ui element and uses the events from the interaction from the user to modify the model. We use the abstract factory pattern to separate libgdx from our view as much as possible.



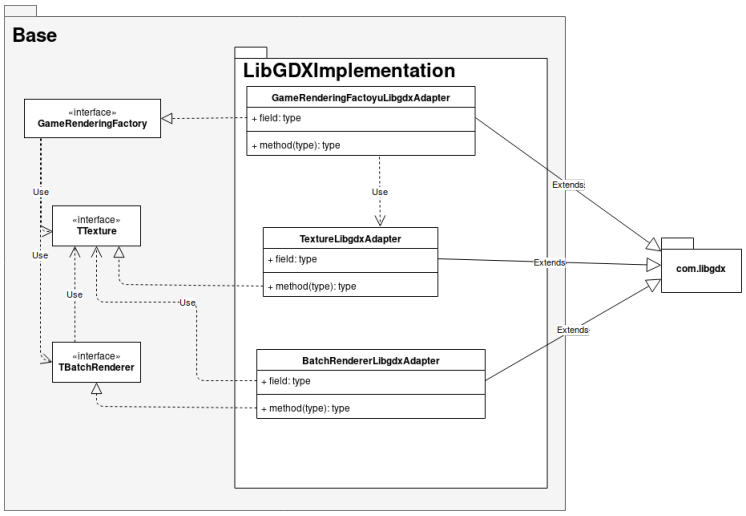
This is the concrete definition of our ui elements.



The **gamerendering** listens to our models entities, and render those who need to be rendred. Of course the model do not know if they need to be rendered, they should not know this. The gamerendering package knows that. We have a few classes that helps with rendering. We use abstract factory pattern to define these.



This is the logic on how we actually render entities.



The concrete implementations in libgdx.

4.2.4 IO

The IO package is responsible for saving and loading the model.

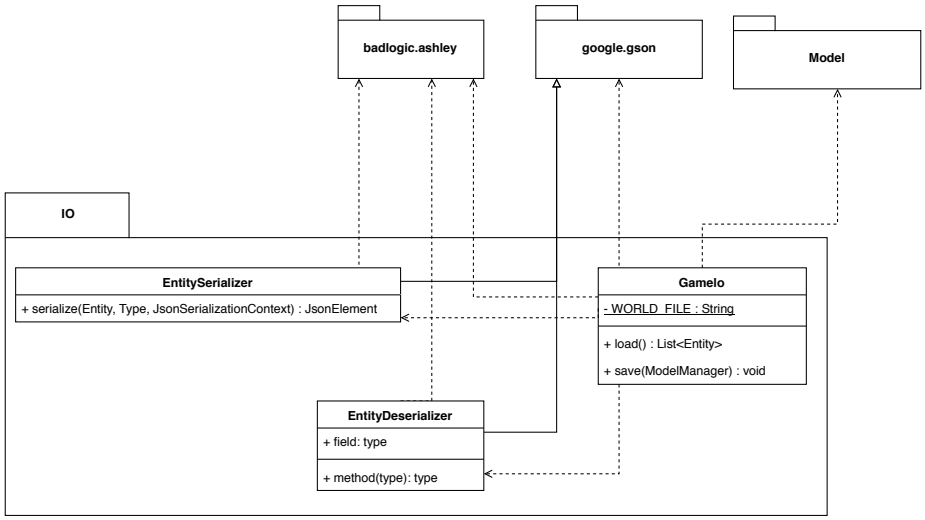
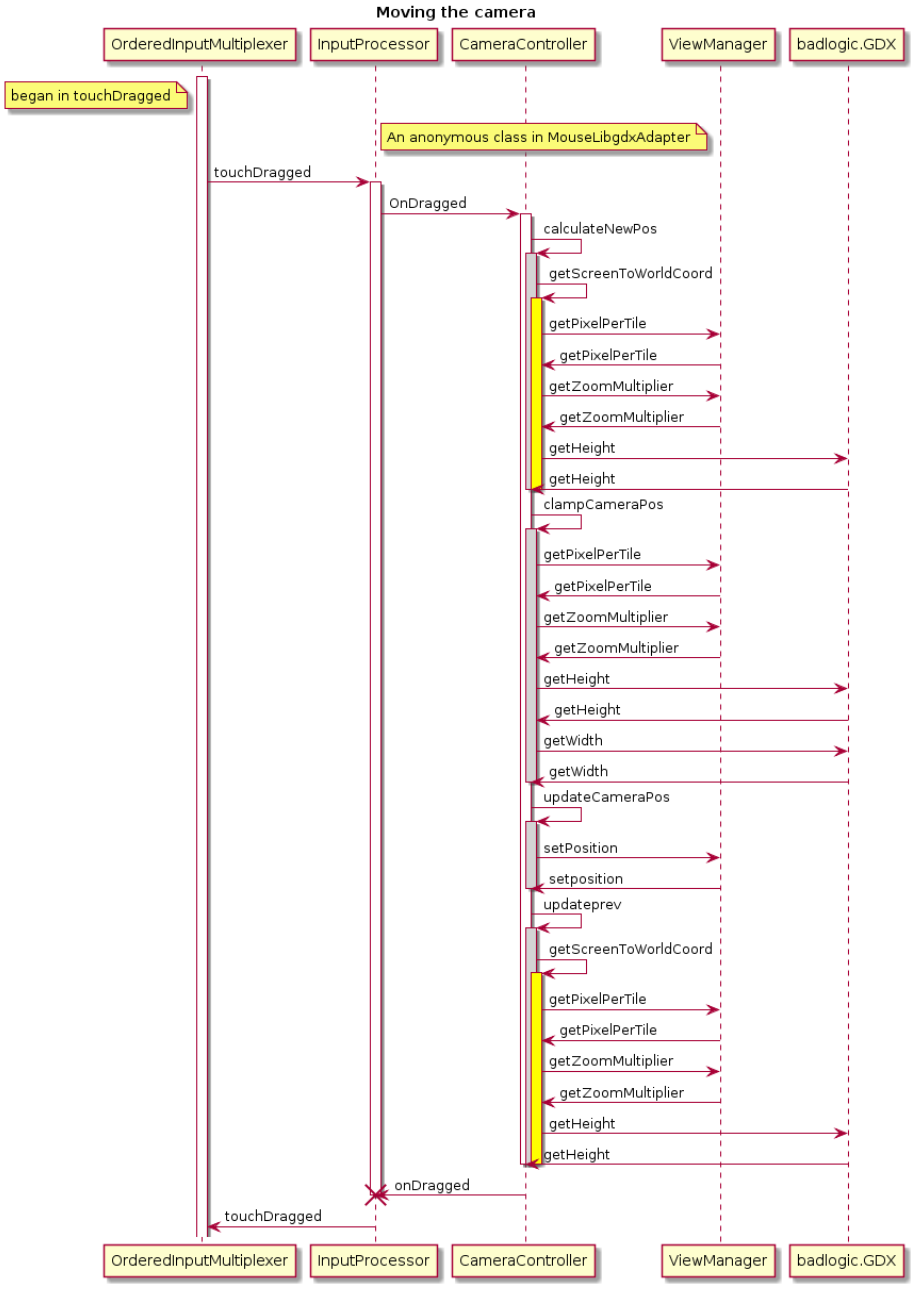
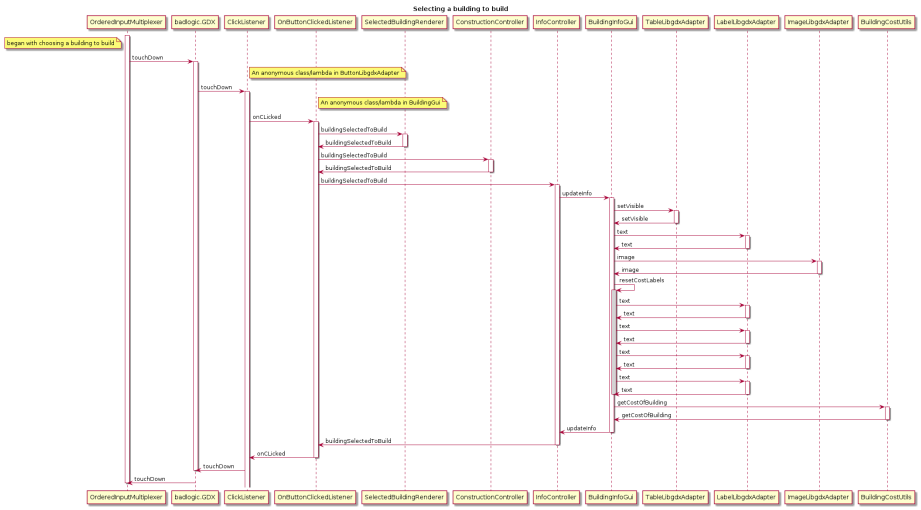


Figure 15: The IO package.

The EntitySerializer formats an entity as a json element, the EntityDeserializer converts a json element back to an entity. The GameIO class takes the model and by using the EntitySerializer and EntityDeserializer converts it to and from a json file.

4.3 Sequence diagram





5 Persistent data management

The application saves almost all of the entities in the model. Some of them, like tile entities, can be regenerated using other information and does not need to be saved. If the size of the world stays the same then there's no problem regenerating the tiles again.

The application uses the GSON library to go through all entities and their components and save them to a json file. When closing the application the world is saved automatically and when starting the application again, the user has the option to either load the previous world.

6 Access control and security

Since the project is a single-user standalone application the only time access control or security should be kept in mind is when the save-file is created. If the user has access to the save-file it is easy to simply change it if they want to cheat. However this is not something that is handled by the application because it doesn't make a big difference since the user would only be cheating against the game and not other players.