

CS 319 - Object-Oriented Software Engineering

Design Report

Syracuse

Supervisor: Uğur Doğrusöz

Group 27

Muhammad Usman

Onur Kulaksızoğlu

Ayşe Kübra Taş

Table of Contents

[1. Introduction](#_uiribfg4jp05)

[1.1 Purpose of the System](#_2h7eprq1uzx7)

[1.2 Design Goals](#_3l1ud67nt43s)

[2. Software Architecture](#_9tpt7ee6bdg7)

[2.1 Subsystem Decomposition](#_xzgxihr70g6t)

[2.2 Hardware/Software Mapping](#_525tilazs31u)

[2.3 Persistent Data Management](#_8ih5a4ab6w7u)

[2.4 Access Control and Security](#_v1wqhwss8cvk)

[2.5 Boundary Conditions](#_7yzhnmtzv74k)

[3.Subsystem Services](#_imiggbe804z7)

[3.1. View Subsystems](#_k92zom2k4fdi)

[3.1.1. Pre-Game Views](#_5wljht3fdta0)

[3.1.2. Play Game Views](#_hgkpah7jvc6o)

[3.2. Controller Subsystems](#_p00itqop4ey5)

[3.2.1. Input Manager](#_yxyw5ydipj25)

[3.2.2. Game Manager](#_8ea7sjip1q78)

[3.2.3. Sound Manager](#_31ga3cgwul7w)

[3.2.4. Persistent Data Manager](#_c1w996eyouba)

[3.3 Model Subsystems](#_j70rhvtri6a2)

[3.3.1. Current Game Memory](#_1f5w381wmwnl)

[3.3.2. Persistent Data](#_3is1ek1kvyf3)

[3.3.3. Game Element Classes](#_hc0rug3n88lt)

[4. Low-level design](#_kyvhin2rrysf)

[4.1. Object design trade-offs](#_lzufipuw3p2g)

[4.2. Final object design](#_xr9jqq5wnthc)

[4.2.1. Views](#_5ddifjkbgnk2)

[4.2.1.1. Pre-Game Frame and Its Related Classes](#_vh85lx6gdscx)

[4.2.1.2. Play-Game Frame and Its Related Classes](#_mxi6u1w27ot7)

[4.2.2. Controllers](#_lxv1ntytzp)

[4.2.2.1.Game Manager](#_rvxjmly9bbpv)

[4.2.3. Models](#_stwkhfuvodi)

[4.2.3.1. Game Element Classes](#_q5o7oxz041gu)

[4.2.3.1.1. Sicilopedia Related Classes](#_jmkuycrgwsxm)

[4.2.3.1.2. Material Classes](#_dz6b5t6q3b7c)

[4.2.3.1.3. People Classes](#_j0dblgkbc9kf)

[4.2.3.1.4. Root Building Class, and its Direct Children](#_dpz2wub2yozj)

[4.2.3.1.5.Work Building Class And its Children](#_chjidhssznfs)

[4.2.3.1.6. Production Building and its Children](#_4qtmp3h0aftb)

[4.2.3.1.7.Civic Classes](#_g9t0yjkl5oi8)

[4.2.3.1.8. Trade Related Classes](#_8xkmvyi0qn6t)

[4.2.3.1.9. Military Organization](#_jfp4496zcpz9)

[4.2.3.1.10. Province Organization](#_ww18yeitnjtw)

[4.2.3.1.11. Faction Organization](#_t7qovl95ucvx)

[4.2.3.1.12.Organization Of Mission Classes](#_hdslg03a97y)

[4.2.3.2.Current Game Memory](#_85j2gnld9btj)

[4.3. Packages](#_9tt11fcofml)

[4.3.1. Views Package](#_1ojypuyc6nai)

[4.3.1.1. Pre-Game Package](#_1xxmfh847wcz)

[4.3.1.2. Play-Game Package](#_enj7e8v1x3az)

[4.3.2. Controllers Package](#_wvhp6xf9vz5k)

[4.3.3. Models Package](#_2uji2uvi7ch)

[4.4. Class Interfaces](#_uj5ig21700mo)

[4.4.1. Pre-Game Frame](#_9gvrfxihhz5c)

[4.4.1.1. Main Menu Panel](#_ogor1ga74362)

[4.4.1.2. Load Game Panel](#_hno6zky2354r)

[4.4.1.3. Settings Panel](#_uxs6176k3nnp)

[4.4.1.4. Trophies Panel](#_wext7nalamm)

[4.4.1.5. Sicilopedia Panel](#_w6dlqp4xe0mz)

[4.4.2. Play-Game Frame](#_19xkiwbmclkz)

[4.4.2.1. In-Game Menu Panel](#_3ov0nx5fio1d)

[4.4.2.2. City Layer Panel](#_6dr4b7m3rtry)

[4.4.2.3. Sicily Layer Panel](#_f5ydv4c5q3v7)

[4.4.2.4. Grand Map Layer](#_vsnsxvn7r15v)

[4.4.2.5. Civics Panel](#_dpfbz7rhxoo9)

[4.4.2.6. Production Storage Panel](#_edypmwtemefk)

[4.4.2.7. Trade Panel](#_3o12p7l2qwft)

[4.4.2.8. Diplomacy Panel](#_kyq9e6i71zir)

[4.4.2.9. Recruitment Panel](#_mu7xryix44w5)

[4.4.2.10. Mercenary Panel](#_of9s327fjm6)

[4.4.2.11. Reports-Notification Panel](#_noivk1uqy3e5)

[4.4.2.12.Population Happiness Panel](#_58yjkttlaerh)

[4.4.2.13.Available Generals Panel](#_hqlo9pb7d4eh)

[4.4.3. Sound Manager](#_nehy6cmp64fv)

[4.4.4.Game Element Classes](#_38jbsnu6wnop)

[4.4.4.1.Material](#_ewk66tqbc3gy)

[4.4.4.2.RawMaterial](#_j6yta28nxkuf)

[4.4.4.3.SeasonalMaterial](#_9oimvfgqyvc7)

[4.4.4.4.ManufacturedMaterial](#_x26pjk54f2th)

[4.4.4.5.Sicilopedia](#_fmw5vlu792vq)

[4.4.4.6.SicilopediaObject](#_81rqapal0g0n)

[4.4.4.7.SicilopediaCategory](#_ili4wn6tcs14)

[4.4.4.8.SicilopediaEntry](#_9alwa7mswbkc)

[4.4.4.9.People](#_3m487uxbn6ek)

[4.4.4.10.FreePeople](#_cemjk29h7fzf)

[4.4.4.11.Slave](#_2iqnaeggpspg)

[4.4.4.12.Proletariat](#_6t48rdtc7sjr)

[4.4.4.13.Aristoi](#_tkdck2on8sgy)

[4.4.4.14.Specifications](#_25fomo38meml)

[4.4.4.15. Building](#_tmlzd05qqfqe)

[4.4.4.16.Housebuilding](#_dt3cmtpsxl9x)

[4.4.4.17.House](#_sfwkb64al0rv)

[4.4.4.18.Mansion](#_95hqneng1trj)

[4.4.4.19.Road](#_oqe6meu1w80h)

[4.4.4.20.Agora](#_tlr24xgytn9k)

[4.4.4.21.WorkBuilding](#_kw29kmmiem5i)

[4.4.4.22.ArcheryRange](#_uv37ydrucltr)

[4.4.4.23.SiegeWorkshop](#_hdvyr2r77ul9)

[4.4.4.24.Library](#_daq44n51yb25)

[4.4.4.25.PublicBath](#_rggb7lxjlltw)

[4.4.4.26.GeneralsHeadquarter](#_9eiqmwrdod5h)

[4.4.4.27.Barrack](#_xmvi2knm5kne)

[4.4.4.28.Courthouse](#_t5jkyu1ai5dc)

[4.4.4.29.Shipyard](#_76h6peetqhf2)

[4.4.4.30.LightHouse](#_cvxlewgbw3g4)

[4.4.4.31.Palace](#_l2vpk4ib7crx)

[4.4.4.32.Theatre](#_3yhwvkg4hnff)

[4.4.4.33.Temple](#_8fwu3dcgi5rh)

[4.4.4.34.Shrine](#_cl938j929q9u)

[4.4.4.35.Academy](#_rygilwszcsxj)

[4.4.4.36.Observatory](#_6u2tbbhy8fyv)

[4.4.4.37.SlaveTrader](#_d1m9l71hv3q3)

[4.4.4.38.CanStore](#_94elq0tfvdra)

[4.4.4.39.MetalMerchant](#_nhlsosf53u54)

[4.4.4.40.GrapeJuiceMerchant](#_kzesexpcdu0t)

[4.4.4.41.Granary](#_bmqgqruerv5k)

[4.4.4.42.OilMerchant](#_u9owm31ieqtw)

[4.4.4.43.SaltMerchant](#_pbd1ny8z22ek)

[4.4.4.44.CheeseMerchant](#_qetnliom3nt)

[4.4.4.45.ProductionBuilding](#_cykwt34u9fxb)

[4.4.4.46.OilPresser](#_5iby03l6s74)

[4.4.4.47.Bakery](#_p4tbmpykmhvd)

[4.4.4.48.GrapeJuiceDistillery](#_rlch101w9wib)

[4.4.4.49.ProductionAndStorageBuilding](#_3bncu9lzjdwr)

[4.4.4.50.WeaponMaker](#_v465ahhzlaes)

[4.4.4.51.FabricMerchant](#_hcj7vz80rg8s)

[4.4.4.52.PigFarm](#_os8dh93qt9on)

[4.4.4.53.Armory](#_vt5n6dr1o29x)

[4.4.4.54.LumberMerchant](#_953rdp4b3hvs)

[4.4.4.55.Butchery](#_ks2oxhf9o49x)

[4.4.4.56.GlassMaker](#_w9cgndi7n8f0)

[4.4.4.57.Fishery](#_73hnvrr7v4du)

[4.4.4.58.Mill](#_a1s0wx2akxse)

[4.4.4.59.MarbleCutter](#_ay71b3wlhqb2)

[4.4.4.60.PotteryMaker](#_3sxsdguyqm5m)

[4.4.4.61.Stable](#_frx895c81fbt)

[4.4.4.62.Jeweler](#_iclp8bfopqhm)

[4.4.4.63.Forge](#_f54c7xhwkc7o)

[4.4.4.64.Tailor](#_2v92nicg2lpl)

[4.4.4.65.TradeOffer](#_senhoxg7uzdl)

[4.4.4.66.TradeShipment](#_kxubzfb4lv6)

[4.4.4.67.Faction](#_ihbl7fsvv5p6)

[4.4.4.68.PlayerFaction](#_2zplh8i81ngo)

[4.4.4.69.AIFaction](#_11dnzbr1wn2j)

[4.4.4.70.ThirdAgeFaction](#_cpgs3qfn4xie)

[4.4.4.71.SicilyFaction](#_r5fchiuiloer)

[4.4.4.72.PeloponnesianFaction](#_kmbhmgj9pohj)

[4.4.4.73.Mission](#_kq0evy263s67)

[4.4.4.74.DeclareWar](#_tsks6bysrsxj)

[4.4.4.75.BuildBuilding](#_kmffr52ckv2i)

[4.4.4.76.StopTrading](#_m0fpsvfyubpl)

[4.4.4.77.RecruitSoldier](#_bh8t3cfypt8w)

[4.4.4.78.CivicChange](#_rt1y5nknvcd5)

[4.4.4.79.ResourceDemand](#_w132p59n47lw)

[4.4.4.80.SendAid](#_bjr6bay6ueuq)

[4.4.4.81.Reward](#_p9almqv9c3vu)

[4.4.4.82.DeadlineReward](#_jpi335dprcbe)

[4.4.4.83.SoldierReward](#_kqyckko5z2bq)

[4.4.4.84.ResourceReward](#_c7lw4san10zw)

[4.4.4.85.DiplomaticOffer](#_juj1sbkgqwht)

[4.4.4.86.Place](#_s7p6kpsbxrxd)

[4.4.4.87.Region](#_5rfri0dt8bz5)

[4.4.4.88.Province](#_mnjw8frpwx1u)

[4.4.4.89.SeaProvince](#_jvob7is7uz50)

[4.4.4.90.LandProvince](#_biad8k5ihmv2)

[4.4.4.91.County](#_5urxknuzl8kn)

[4.4.4.92.CityProvince](#_mo8dqkhvf5bk)

[4.4.4.93.Civic](#_yhlmuyhkdenn)

[4.4.4.94.Force](#_tf3u1798r6mx)

[4.4.4.95.Garrison](#_tx72097s7z65)

[4.4.4.96.Navy](#_p85vvadrnuzw)

[4.4.4.97.Army](#_9k8rgh72oee6)

[4.4.4.98.MercenaryArmy](#_8y0o7lbxn04q)

[4.4.4.99.General](#_9fhefu44dizg)

[4.4.4.100.Unit](#_j5wrn111g3a2)

[4.4.4.101.Ship](#_566hni6yg8dx)

[4.4.4.102.LandUnit](#_y6h2ghiof5xz)

[4.4.5. Current Game Memory](#_logxs6sto76o)

[4.4.6. Game Manager](#_l3ah7vvowgho)

[5. Glossary & references](#_1c8xq16kepl0)

# 

# 1. Introduction

## 1.1 Purpose of the System

Syracuse is a turn based city/state building simulation/strategy game inspired by the game Tropico. In this game the player takes control of the city Syracuse during the antiquity. The main goal of the player is to defend, expand and develop his/her city. The game is designed to be user friendly and easy to learn. It get difficult as the player progresses in order to keep the user interested. The main focus of the game is its gameplay which means that some other things like graphics are not that good as some other games of the same genre. But this game contains loads of interesting missions and quests to keep the player engaged for hours.

## 1.2 Design Goals

Usability/ User friendliness: We have put a great emphasis on making the game really easy to use.The user can easily switch between different layers to like the city layer or Sicily layer to get a better understanding of what's going on and make decisions easily. Apart from the we have a turn based system which means that the player can play the game at any pace and whenever they want and once he completes all the missions, he can ask to go onto the next level/age. All these features make the game really easy and friendly to use.

Ease of learning: Often strategy games contain a lot of stuff going on at the same moment and the player has to look at all of them and makes decisions accordingly. It's the same with our game. That’s why we have the Sicilopedia. It is an encyclopedia which contains information about each and every aspect of the game and the player can refer to it whenever he is facing any difficulty. So this feature helps the user in learning the ins and outs of this game.

Maintainability/ Modifiability: We have chosen to use Java as the programming language for our game. As it is pretty easy to use and maintain than some other languages like C++. So that’s why our game will be really easy to maintain and update whenever we feel the need to do it.

**TradeOffs:**

Rapid Development vs Eye Candy: We have chosen to focus on core gameplay elements of the game and not putting very much emphasis on the graphics so that we can easily and fastly code the game and put it out to be used by the users.

Efficiency vs Portability: We are releasing this game for Windows only so that we can make it as fast and efficient as possible and not worry about making it cross platform as it might require us to implement some features that might overall decrease the game performance.

Cost vs Robustness: We are choosing to reduce the cost by implementing everything really fastly and using just the java native libraries. So the game might not be as fast and responsive as some other strategy games. Plus we have chosen not to focus much on graphics, we are going to reduce the cost by using simple/free graphical tools instead of buying a professional graphical design tool.

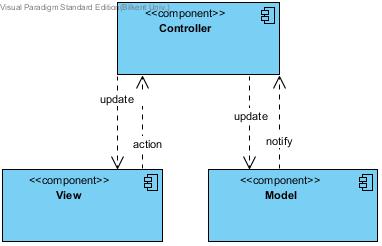
Speed vs Space: As our game doesn't use heavy graphical elements, the loading and saving of game will be pretty fast as there will be not much of the data to be read or written and this in turn will also reduce the space requirements for our game.

# 2. Software Architecture

In this section, we will decompose our system into subsystems to make it more understandable and to facilitate our job during implementation . We will also explain which hardware and software tools are needed and how to manage database system. Additionally, access control and security issues will be mentioned and boundary conditions will be examined under three subsections.

## 2.1 Subsystem Decomposition

Our game, as a strategy game, includes many screens and pretty much logic behind these screens which need to be considered separately. For this screen-logic separation we decide to use MVC pattern which provides minimizing coupling between main subsystems and maximizing cohesion within a subsystem that is while view subsystem is handling all user interface stuffs, model subsystem is dealing with background logic issues and controller subsystem interacts with user while updating model and view subsystems and receiving actions from view.



There are subsystems of model, view and controller components in the diagram below. Relationships between these subsystems are also defined in the diagram.

## 

## 2.2 Hardware/Software Mapping

We choose Java SE platform that uses object-oriented Java language to implement our game with its Java development Kit (JDK) and Java Runtime Environment (JRE). The packages we mainly use are AWT package which handle basic GUI operations, swing package which handles widgets and util package that contains many framework, event model etc. The only hardware need for our project is a keyboard and and a mouse.

## 2.3 Persistent Data Management

In the game, we should keep data to save game and for achievements. For this purpose, a complex database system is not needed but instead we can use simple text files. Text files which keep saved games and achievements would not be used in the current game. These are just for the player so that he can reach when he wants. Also, we are planing to use some gifts and music files which can be used in the current game.

## 2.4 Access Control and Security

Our game is a single-player game which does not require any network connection. Player needs to download it once and then, only people who can access the game are those who can access the computer in which our game exist. Therefore, there would not be a security problem as long as the computer is in safe.

## 2.5 Boundary Conditions

**Initialization**

The game will start after clicking on the icon on the desktop or any place in the computer. The extension of this icon will be .jar.

**Termination**

Player can leave the game by clicking “Quit Game”. Then we would ask if the player wants to save his played game. If he chooses to save, game will be saved and the player can continue later. There no need to pause game because game is a term-based game.

**Error**

For any reason, if game gives any error, game will not be saved. Player can start a new game or he continue with previously saved games. Only current games can be affected by the errors since we are saving played games in text files.

# 

# 

# 

# 

# 

# 

# 

# 3.Subsystem Services

The parts below in this section describes the brief duties of the each subsystem.

## 3.1. View Subsystems

Because of our MVC design architecture all the subsystems that handles the interface are collected in this larger subsystem.

### 3.1.1. Pre-Game Views

The pre-game views is the subsystem which includes the user interface classes which are showed on the screen before game starts. This subsystem receives sicilopedia entries from the game elements classes.

### 3.1.2. Play Game Views

The play-game views is the subsystem which includes the user interface classes which are showed on the screen when game starts. All graphic related stuffs are handled by play-game views subsystem. This subsystem receives user inputs and sends them to the input manager and it is modified by Game Manager subsystem which is one of the controller components.

## 3.2. Controller Subsystems

Controller subsystems in our game handles all data accesses and modifications, and also modifying the values in user interfaces. Controller subsystems also handles game logic, sound effects in the game, and input management.

### 3.2.1. Input Manager

Input Manager is responsible for taking inputs from the player and sending them appropriately to the Game Manager, Game Manager calls the appropriate functions according to these inputs.

### 3.2.2. Game Manager

Game Manager is responsible for handling all the core game logic, modifying the current game data and the game graphics while also doing them according to the user inputs. Game Manager also calls appropriate functions from the subsystems of Sound Manager and Persistent Data Manager, again according to the current game situation and user inputs. This is the most central and important subsystem of our game.

### 3.2.3. Sound Manager

Sound Manager subsystem provides an interface to the Game Manager for playing sounds and musics dependent on the game’s current situation. Other than that this subsystem has no function.

### 3.2.4. Persistent Data Manager

Persistent Data Manager subsystem provides an interface to the Game Manager to be used when player wants load-save games and trophies. Persistent Data Manager subsystem handles all the things related to the data in the hard disk and provides modularity in our system design.

## 3.3 Model Subsystems

Model subsystems in our design represent both the data that is persistent on the disk like saved games and the data on the memory about the current game like maps etc.

### 3.3.1. Current Game Memory

This subsystem’s purpose is to hold the memory objects special to this session of the game, like the player’s settings, his current diplomatic relationships, buildings in his city, which age he is in etc. This memory is modified-created by Game Manager subsystem and is again only accessed by this subsystem. When a game is saved only this part will be stored as persistent data.

### 3.3.2. Persistent Data

Persistent Data subsystem represents the data that are saved in the hard disk in our game, the data for the trophies and the saved games are part of this sub-file system. Persistent data can only be accessed and modified by the Persistent Data Manager subsystem when it’s ordered by the Game Manager.

### 3.3.3. Game Element Classes

Game Element Classes” model subsystem includes the data that are not unique to only one session of the game and are also not persistent in the hard disk. Objects for Sicilopedia entries, individual building features, unit features, agendas of the other factions and many other data fragments required by the game are organized in this model subsystem. To give an example between the difference of previous parts and this part: this data part holds the information about which raw resource is required for manufacturing a good that is same in every game, while the “Current Game Memory” subsystem holds the information about the current resources owned by the player in the city.

# 4. Low-level design

## 4.1. Object design trade-offs

### 4.1.1. Functional decomposition vs less code:

In our object design we have assigned a separate function for each operation. Though it will increase the no. of lines of code but maintaining and understanding our code will be easier due to the decomposition.

### 4.1.2.Graphics vs Space requirement :

Since we are saving the objects directly to the hard drive for save operations, we chose to reduce the amount of graphics as much possible, so that the size of each object to be save is small.

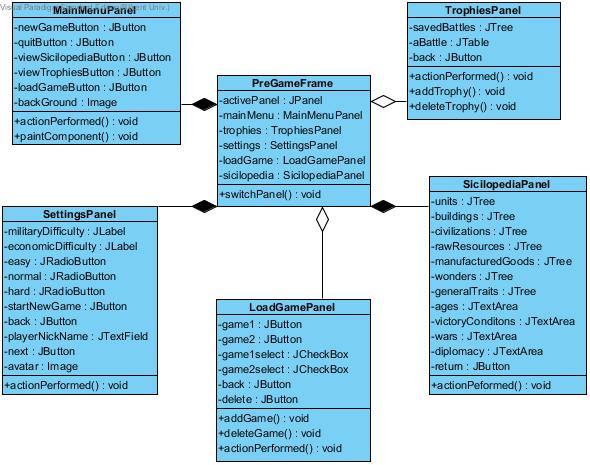
## 4.2. Final object design

### 4.2.1. Views

Game views are showed under two frames: pre-game frame and play-game frame. These frames and their related classes are illustrated below.

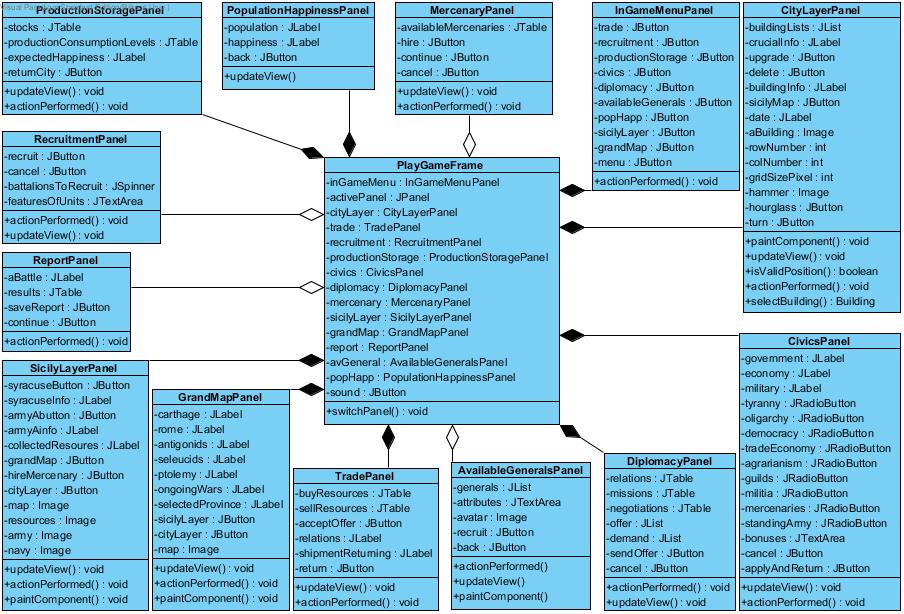
#### 4.2.1.1. Pre-Game Frame and Its Related Classes

Pre-game frame is related with 5 classes which are for the panels showed before the game. Player can see trophies, do settings, choose a load game and see the sicilopedia on that frame.



#### 4.2.1.2. Play-Game Frame and Its Related Classes

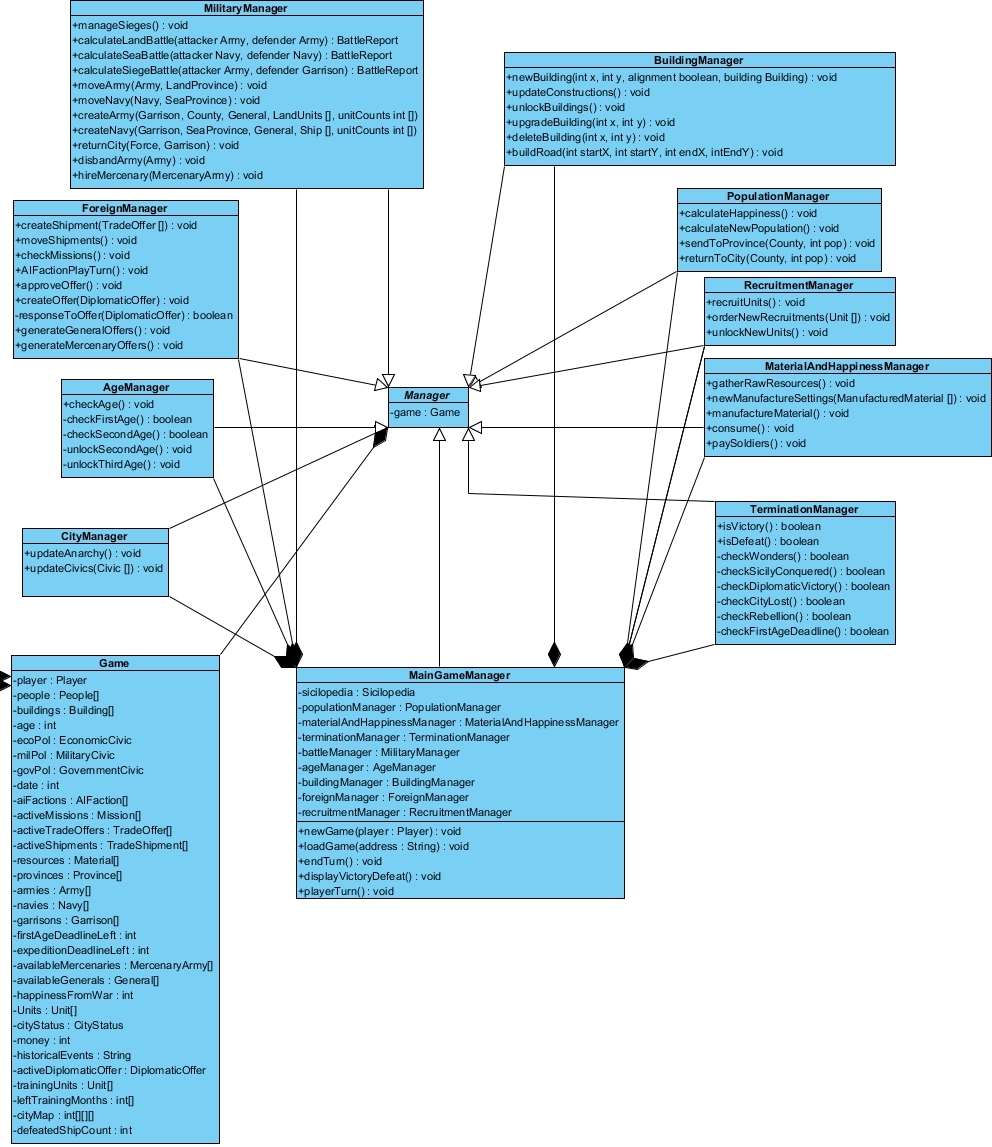
Play game frame is related with 13 classes as illustrated above. All these classes are part of playing the game. Basicly, game will be played on that frame and on the active panel.



### 4.2.2. Controllers

#### 4.2.2.1.Game Manager

Game Manager subsystem consists of 10 manager classes that all inherit the “Manager” class which means all manager classes have direct access to the “Game” object. Main Game Manager is the class that responsible from delegating all the things asked by the user to the other manager classes.



#### 4.2.2.2. Sound Manager

This subsystem enable the game to have background sound. It has one class and inside this classes there is private soundListener class.

#### 4.2.2.3. Input Manager

We will define all our different axes input and game actions for our project in the input manager.

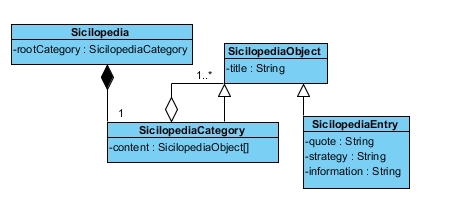
### 4.2.3. Models

Game’s model subsystems can be divided into three, one is the “Current Game Memory” which holds the information related to current session of the game, and the second one is “Game Element Classes” which holds all the other model classes used by the game. The last one “Persistent Data” does not consist of any classes, but represents the games and trophies that are saved in the hard disk.

#### 4.2.3.1. Game Element Classes

Since there are too many classes in this section, to make it look nicer classes will be divided into groups, from the base-provider classes to more specific classes.

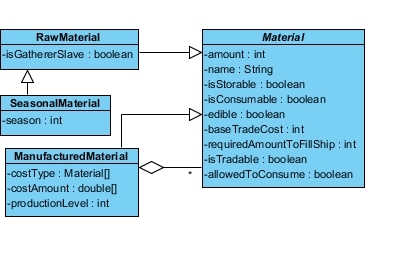
##### 4.2.3.1.1. Sicilopedia Related Classes



Sicilopedia objects are organized in these classes as illustrated above.

##### 4.2.3.1.2. Material Classes

Materials-Resources are organized in the game as following:



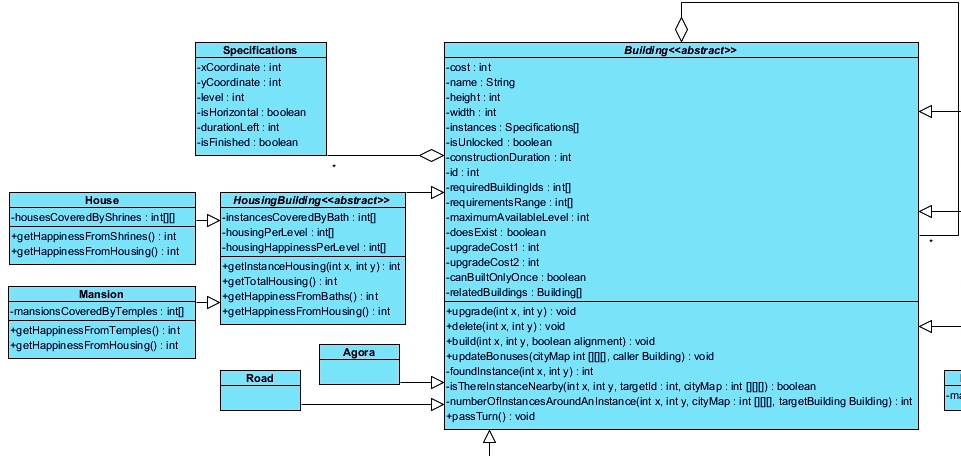
##### 4.2.3.1.3. People Classes

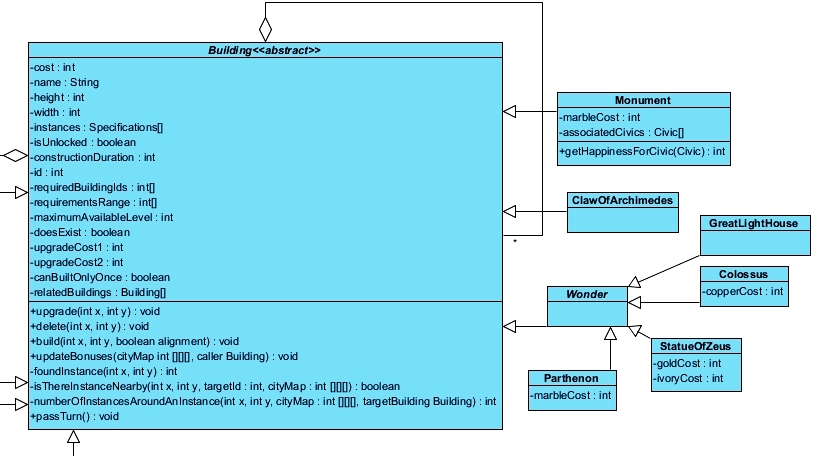
Three types of people in the game are organized in the following way:



##### 4.2.3.1.4. Root Building Class, and its Direct Children

The following classes show our base building class, which all other buildings are derived, additionally some of its extenders are shown like wonders and housing buildings:

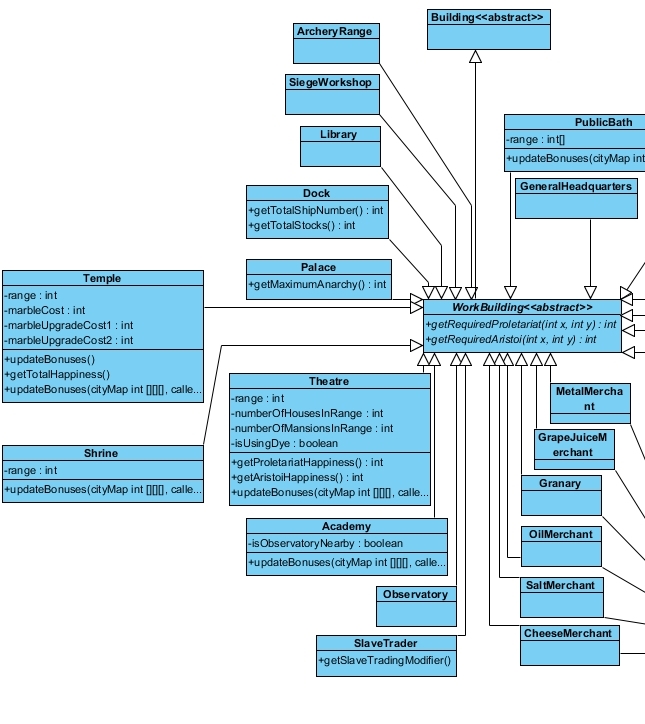


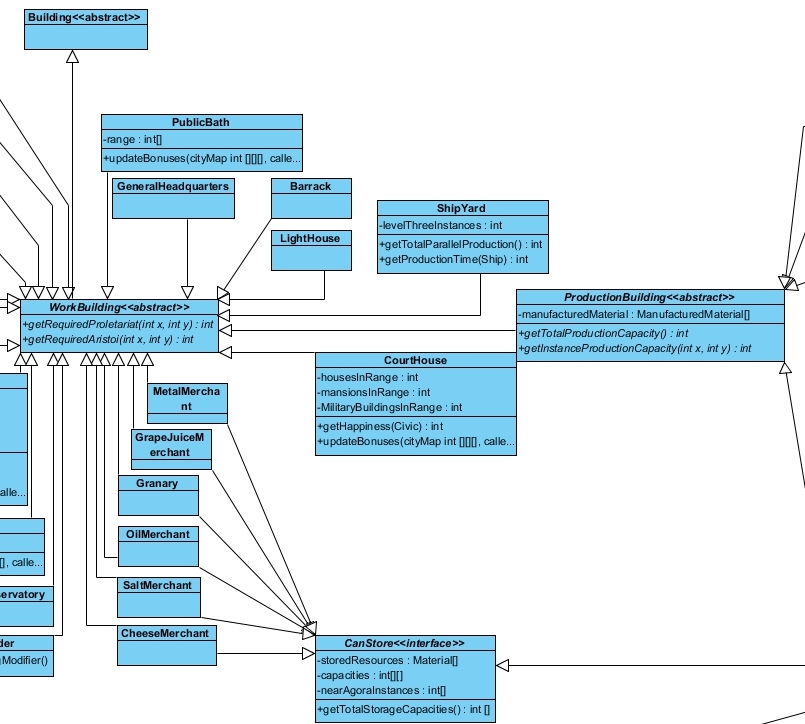


##### 

##### 4.2.3.1.5.Work Building Class And its Children

The work building is a class that extends buildin class, also it is a parent of leaf and other parent classes, like production building or theatre etc.





##### 4.2.3.1.6. Production Building and its Children

Production Building is an extender of Work Building class, Production and Storage Building class is an extender of Production Building class which also implements the Storage interface. Buildings related to these classes are shown below:

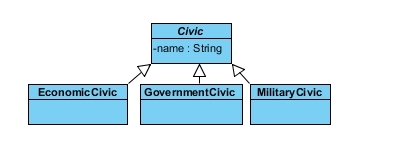


Bottom part:



##### 4.2.3.1.7.Civic Classes

Civic Classes are organized in the following way, since civics are handled by manager classes the model classes are not very complicated:



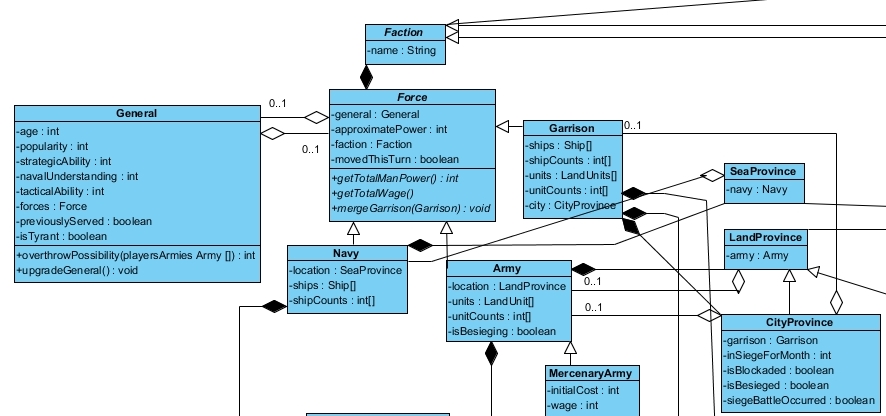
##### 4.2.3.1.8. Trade Related Classes



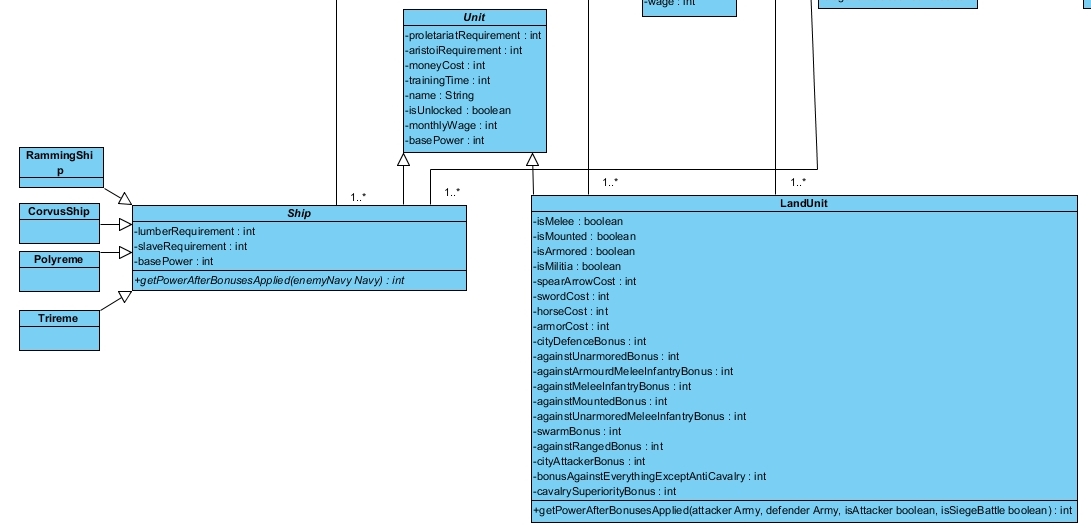
Diagram depicts how the factions are initially organized, what do trade offers, provinces and shipments include.

##### 4.2.3.1.9. Military Organization

Diagram shows how the military in the game organized:

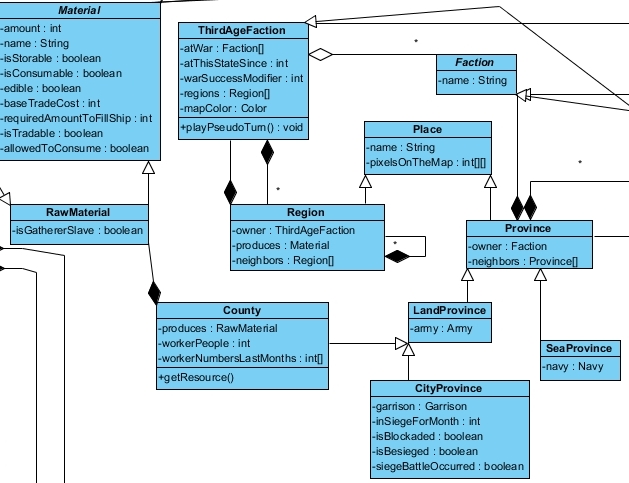


Arrows got messy in the above diagram but the diagram basically tells that, every force has a faction, a force can be an army a garrison, or a navy. A force may have a general and a general may have a force. Every army is in a land province, every garrison is in a city province and every navy is in a sea province. A city province may have a garrison, and may have an army at the same time, a land province and a sea province can have at most one force on them. Every Mercenary army is an army.



The bottom part of the military section shows how units are organized in the game, forces in the game consists of these units.

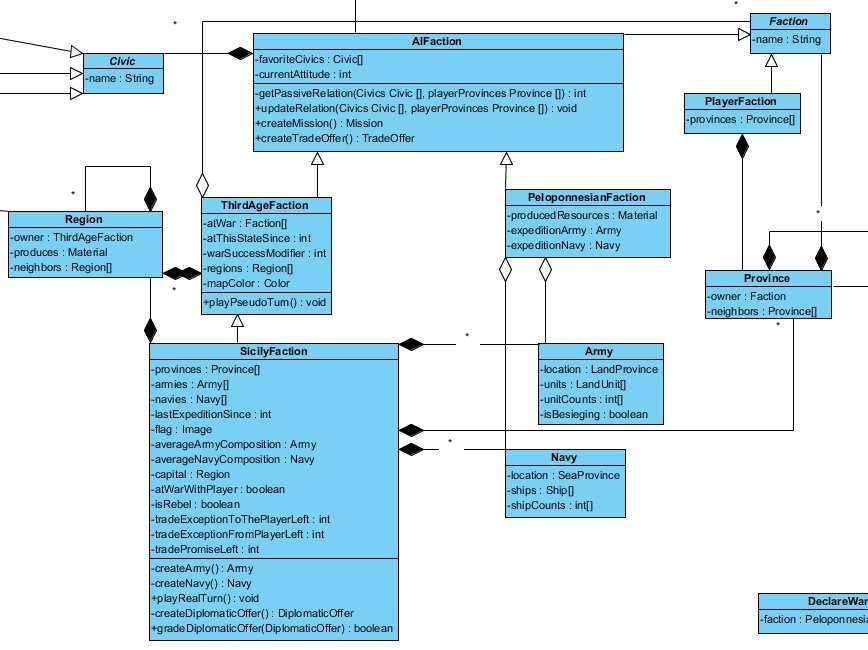
##### 4.2.3.1.10. Province Organization

Again this diagram is pretty messy because of the relations, but here is the diagram:

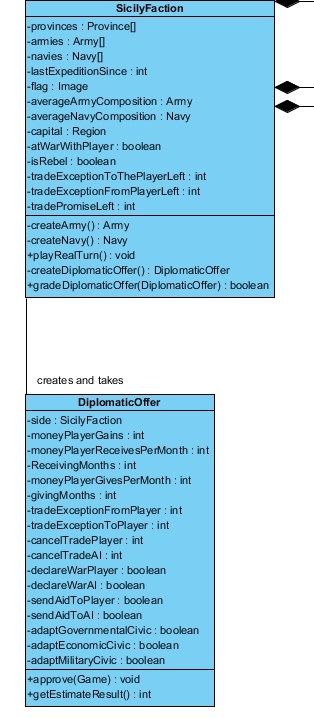
Center of the diagram is the place object, where regions and provinces are originated, provinces are the divided places in the Sicily Map, while the regions are in the Grand Map. Provinces can have forces on them and produces raw materials, while regions don’t have forces on them and can produce all types of materials.

##### 4.2.3.1.11. Faction Organization

Below diagram shows how the factions and their related classes are organized in the game:

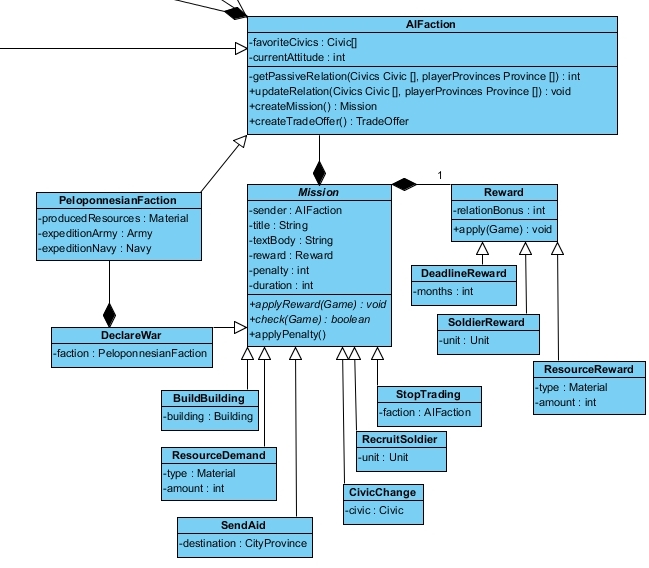


Below diagram shows the DiplomaticOffer class which is created by or sent to SicilyFaction objects:



##### 4.2.3.1.12.Organization Of Mission Classes

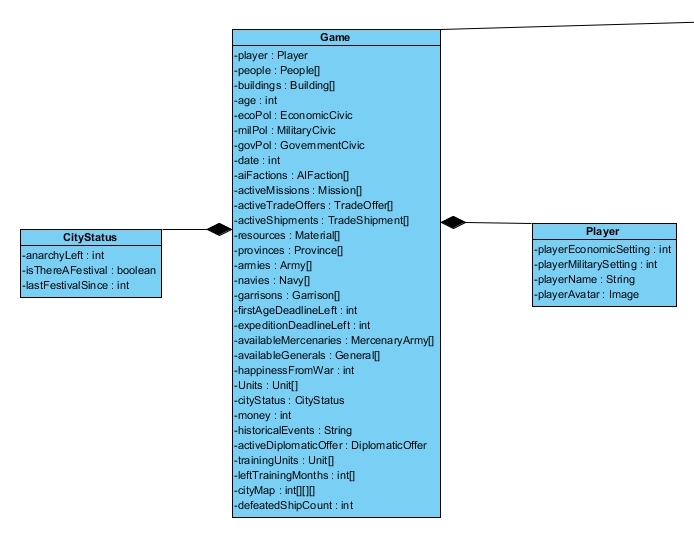
Mission class objects are created by Faction objects, and each of them has a reward. To not make diagram too messy, the mission related objects like “building”, “unit”, or “civic” are omitted.



There are 7 types of missions that can be given to the player by AIFactions, and 4 types of rewards that can be gained by player(including just the bonus relation rewards).

#### 4.2.3.2.Current Game Memory

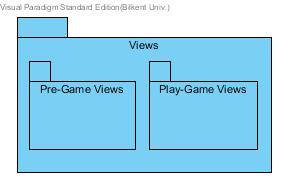
Current Game Memory subsystem actually consists of a single object called “Game”. It is an object that all the information related to the current session of the game is hold, its attributes consist of instances of “Game Element Classes” but again for simplicity all the aggregations made by this class is omitted. Also because of time issues, some more detailed attributes are missing from the Game object, since the manager classes will sometimes need the individual access to the buildings like monuments, yet current attributes does not provide this without casting of the “Building” array. Or while Army[] array is useful for calculating total wages of all armies, manager classes won’t be able to differentiate a mercenary army and a normal army for rebellion purposes. So to speak, during implementation this design will be improved to allow more polymorphism and encapsulation.



## 4.3. Packages

### 4.3.1. Views Package

Views package includes all GUI related classes and has two subpackages: Pre-view package and Play-game package.



#### 4.3.1.1. Pre-Game Package

Pre-game package includes a frame class and 5 panel classes. Frame class is composed and aggregated by the panel classes.

#### 4.3.1.2. Play-Game Package

Play-game package includes a frame class 13 panel classes. Frame class is composed and aggregated by the panel classes.

### 4.3.2. Controllers Package

Controllers package has 3 more subpackages as illustrated below. These packages are responsible for input, game and sound management.

### 

#### 4.3.2.1. Input Manager Package

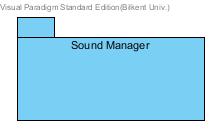
We will define all our different axes input and game actions for our project in the input manager.

#### 4.3.2.2. Game Manager Package

All classes related with game management are in this package. This package has 12 classes which will be implemented to control the game.

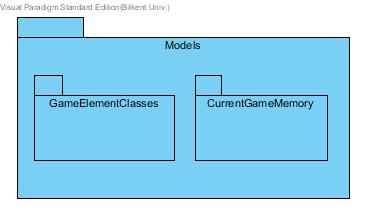
#### 4.3.2.3. Sound Manager Package

This package enabled game to have background sound and only has one class with its private soundListener class.



### 4.3.3. Models Package

Models package has 2 subpackages which are game element classes package and current game memory package. Our entity classes will be in these packages.



#### 4.3.3.1. Game Element Classes Package

Most of the entity classes are in that package. These classes represents general features of the game that means all of composed game will use these classes.This package has more than 100 classes.

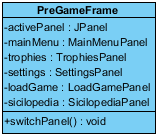
#### 4.3.3.2. Current Game Memory Package

In that package we have 3 classes: Game, Player and CityStatus. These classes are used for the current played game.

## 4.4. Class Interfaces

### 4.4.1. Pre-Game Frame

This class will be extended by JFrame and it will be the frame that player faces when he first clicks on the game icon.



**Attributes**

**JPanel activePanel:** Active panel is the panel which connects with the player on the pre-game frame.

**MainMenuPanel mainMenu:** Main menu panel will be showed on pre-game frame. When active panel is main menu panel, this attribute will be used.

**TrophiesPanel trophies:** Trophies panel will be showed on pre-game frame. When active panel is trophies panel, this attribute will be used.

**SettingsPanel settings:** Settings panel will be showed on pre-game frame. When active panel is settings panel, this attribute will be used.

**LoadGamePanel loadGame:** Load game panel will be showed on pre-game frame. When active panel is load game panel, this attribute will be used.

**SicilopediaPanel sicilopedia:** Sicilopedi panel will be showed on pre-game frame. When active panel is sicilopedia panel, this attribute will be used.

**Constructor**

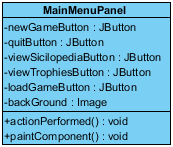
Panels will be composed and initial active panel will be defined as main menu panel.

**Methods**

**switchPanel:** This method enables changing active panel according to player action.

#### 4.4.1.1. Main Menu Panel

Main Menu Panel has only 5 buttons and a background image.



**Attributes**

**JButton newGameButton:** This jbutton enables player to go settings panel.

**JButton viewTrophiesButton:** This jbutton enables player to go trophies panel.

**JButton loadGameButton:** This jbutton enables player to go load game panel.

**JButton quitButton:** This jbutton enables player to close the pre-game frame.

**Image** **backGround**: This image attribute is used to put a background image under buttons.

**Constructor**

Panel will be composed under constructor.

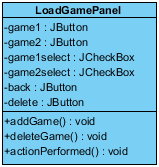
**Methods**

**void actionPerformed:** This method handles all button actions.

**void paintComponent:** Under this method, the background image will be drawn.

#### 4.4.1.2. Load Game Panel

This class enables player to join to the previously saved games.



**Attributes**

Initially only attribute that game will have is delete JButton.

**JButton back:** This jbutton enables player to go main menu.

**Constructor**

The back button will be composed.

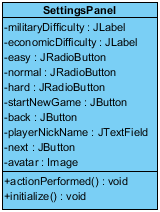
**Methods**

**void addGame():** When player saves some games, this function adds buttons and check boxes to the load game panel. After first game, this function also forms a delete button on this panel.

**void actionPerformed:**Player can click on one of load game button and continue to a load game. After this click, pre-game frame will be closed and play-game frame will be opened. This function handles all these events.

**void deleteGame:** Player can also delete a load game by marking check boxes under load games and then clicking on delete button. This function is responsible for delete button function.

#### 4.4.1.3. Settings Panel



**Attributes**

**JLabel militaryDifficulty:** This is a jlabel attribute of military difficulty on settings panel.

**JLabel economicDifficulty**: This is a jlabel attribute of economic difficulty on settings panel.

**JRadioButton easy:** This is a radio button of easy level.

**JRadioButton normal:** This is a radio button of normal level.

**JRadioButton hard:** This is a radio button of hard level.

**JTextField nickname:** Player can write his nick name in this text field.

**Image avatar:** There will be avatar attributes that player can choose. This image will be on the next button. Number of avatars is determined during implementation.

**JButton next:** This button enables player to move next avatar.

**JButton startNewGame:** This button enables player to start a new game.

**JButton back:** Player can also go back to main menu with this button.

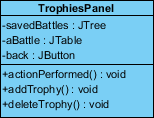
**Constructor**

All buttons, radio buttons, text field and images will be formed under constructor. Initial view of the settings panel will be composed. As default, normal radio buttons will be marked and in text field, nickname will be written. Also, the first avatar will be on the next button.

**Methods**

**void actionPerformed:** This methods handles all button events. When player clicks on new game button, a new game starts on play-game frame while pre-game frame is closed.

#### 4.4.1.4. Trophies Panel



**Attributes**

**JTree savedBattles:** This attribute keeps saved battles in a list which can be clicked.

**JTable aBattle:** This attribute keeps a saved battle on a table.

**JButton back:** This button enables player to go back main menu.

**Constructor**

Initial view of the Trophies panel is composed under constructor.

**Methods**

**void actionPerformed:** Saved battles tree and back button events will be handled under this method.

**addTrophy:** This method will update the saved battles tree and add new tables.

**deleteTrophy:** This method will update the saved battles tree and delete a table.

#### 4.4.1.5. Sicilopedia Panel

##### 

**Attributes**

**JTree units:** This attribute keeps units in a list which can be clicked.

**JTree buildings:** This attribute keeps buildings in a list which can be clicked.

**JTree civilizations:** This attribute keeps civilizations in a list which can be clicked.

**JTree manufacturedGoods:** This attribute keeps manufactured goods in a list which can be clicked.

**JTree wonders:** This attribute keeps wonders in a list which can be clicked.

**JTree generalTraits:** This attribute keeps general traits in a list which can be clicked.

**JTextArea ages:** Ages and their features will be written on this text area.

**JTextArea victoryConditions:**Victory conditions and topics related with victories will be written on this text area.

**JTextArea diplomacy:** Diplomacy and topics related with diplomacy will be written on this text area.

**JTextArea wars:** Wars and their features will be written on this text area.

**JButton return:** This button enables to go back main menu.

**Constructor**

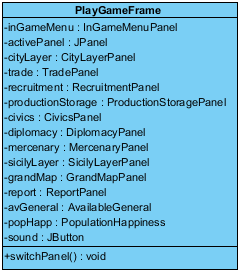
All attributes will be composed under construction.

**Methods**

**void actionPerformed:** All button events will be handled under this method.

### 4.4.2. Play-Game Frame

This frame will be extended by JFrame and when player clicks on new game button or chooses a load game, this frame will be opened.



**Attributes**

**InGameMenuPanel inGameMenu:** There will be this in game menu on the top of the play game frame that enables player to switch on another panel.

**JPanel activePanel:** Active panel is the panel which connects with the player on the play-game frame under in game menu panel.

**CityLayerPanel cityLayer:** City layer panel will be showed on play-game frame. When active panel is Trade panel, this attribute will be used.

**TradePanel trade:** Trade panel will be showed on play-game frame. When active panel is Trade panel, this attribute will be used.

**RecruitmentPanel recruitment:** Recruitment panel will be showed on play-game frame. When active panel is recruitment panel, this attribute will be used.

**ProductionStoragePanel productionStorage:** Production storage panel will be showed on play-game frame. When active panel is production storage panel, this attribute will be used.

**CivicsPanel civics:** Civics panel will be showed on play-game frame. When active panel is civics panel, this attribute will be used.

**DiplomacyPanel diplomacy:** Diplomacy panel will be showed on play-game frame. When diplomacy panel is city layer panel, this attribute will be used.

**MercenaryPanel mercenary:** Mercenary panel will be showed on play-game frame. When active panel is mercenary panel, this attribute will be used.

**SicilyLayerPanel sicilyLayer:** Sicily layer panel will be showed on play-game frame. When active panel is Sicily layer panel, this attribute will be used.

**GrandMapPanel grandMap:** Grand map panel will be showed on play-game frame. When active panel is grand map panel, this attribute will be used.

**ReportPanel report:** Report panel will be showed on play-game frame. When active panel is report panel, this attribute will be used.

**AvailableGeneral avGeneral:** Available general panel will be showed on play-game frame. When active panel is available general panel, this attribute will be used.

**PupulationHappiness popHapp:** Population happiness panel will be showed on play-game frame. When active panel is population happiness panel, this attribute will be used.

**Constructor**

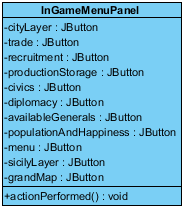
Panels will be composed and initial active panel will be defined as city layer panel.

**Methods**

**switchPanel:** This method enables changing active panel according to player action.

#### 4.4.2.1. In-Game Menu Panel

This class will be extended with JPanel.



**Attributes**

**JButton trade:** This button enables player to switch on trade panel.

**JButton recruitment:** This button enables player to switch on recruitment panel.

**JButton productionStorage:** This button enables player to switch on production storage panel.

**JButton civics:** This button enables player to switch on civics panel.

**JButton diplomacy:** This button enables player to switch on diplomacy panel.

**JButton availableGenerals:** This button enables player to switch on available generals panel.

**JButton popHapp:** This button enables player to switch on population happiness panel.

**JButton sicilyLayer:** This button enables player to switch on sicily layer panel.

**JButton grandMap:** This button enables player to switch on grand map panel.

**JButton menu:** This button enables player to see a small window which includes “save game” and “continue”. If player clicks on save game he will go to the main menu. For other option player will continue the game.

**Constructor**

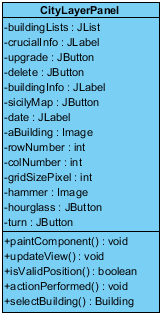
Initial view of the in game menu panel will be composed and buttons will be formed.

**Methods**

**actionPerformed:** All button events will be handles by this method.

#### 4.4.2.2. City Layer Panel

This class will be extended with JPanel.



**Attributes**

**JList buildingList:** From this clickable list, player can choose a building to construct.

**JLabel crucialInfo:** There will be some crucial information about game like available housing for each kind of population on this label.

**JButton upgrade**: This button enables buildings to be upgraded.

**JButton delete:** Player can delete a building using this button.

**JLabel buildingInfo:** When player selects a building, this label will occur and shows information about the selected building.

**JButton sicilyMap:** There will be a button to move to the sicily layer.

**JLabel date:** On this label, player will see the date.

**Image aBuilding:** There will be images for each building. Actually we have this attribute 51 times for different buildings but since it is too many and no need to show all of them here we have one attribute as sample.

**int rowNumber:** Number of rows on the grid of city layer will be defined with this attribute.

**int colNumber:** Number of columns on the grid of city layer will be defined with this attribute.

**int gridSizePixel:** Size of pixels will be defined with this attribute.

**Image hummer:** This image will be used when a building is constructing.

**JButton hourglass:** Player can end turn by clicking this button.

**JButton turn:** When player selects a building this button will occur and enables a building to be turned.

**Constructor**

Constructor will compose initial city layer view by forming all attributes above.

**Methods**

**void paintComponent:** All images will be drawn under this method.

**Building selectBuilding:** This method returns building when player clicks on a building.

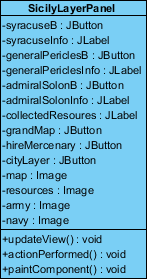
**boolean isValidPosition:** This method returns a boolean when player tries to construct a building on the grid map of city layer.

**actionPerformed:** This method handles all button events.

**updateView:** This method update the city layer according to player actions.

#### 4.4.2.3. Sicily Layer Panel

This class will be extended with JPanel.



**Attributes**

**JButton syracuseButton:** This button enables to show syracuse info label.

**JLabel syaracuseInfo:** Total force and number of units are written on this label.

**JButton armyAbutton:** This button enables to show armyA info label. In the game there will be more army so number this attribute will be larger. ArmyA is just a sample.

**JLabel armyAinfo:** Total force and number of units of armyA will be written on this label. Again, there will be more armies and this is a sample army illustration label.

**JLabel collectedResource:** Player will see the information about collected resources on this panel.

**JButton grandMap:** This button enables player to move to the grand map.

**JButton hireMercenary:** This button enables player to move to the mercenary screen.

**JButton cityLayer:** This buttonenables player to move to the city layer panel.

**Image map:** This image is the image of the sicily map.

**Image resourceA:** This image is the image of a resources. There will be many resources and this is a sample.

**Image armyA:** This is the image of an army. There might be many army. This is a sample army image.

**Image navy:** This is the image of the navy. There is one type navy image.

**Contructor**

Initial sicily screen will be composed under constructor with the some defined attributes above like buttons and labels.

**Methods**

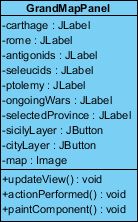
**void paintComponent:** Images will be drawn under this method.

**void actionPerformed:** Button actions will be handled under this method.

**void updateView:** According to player actions, this method update the sicily panel.

#### 4.4.2.4. Grand Map Layer

This class will be extended with JPanel.



**Attributes**

**JLabel carthage:** There will be resources that Carthage has on this panel.

**JLabel rome:** There will be resources that Rome has on this panel.

**JLabel antigonids:** There will be resources that Antigonids has on this panel.

**JLabel seleucids:** There will be resources that Seleucids has on this panel.

**JLabel ptolemy:** There will be resources that Ptolemy has on this panel.

**JLabel ongoingWars:** There will be information about ongoing wars on this panel.

**JButton sicilyLayer:** Player can switch on sicily layer by clicking this button.

**JButton cityLayer:** Player can switch on city layer by clicking this button.

**Image map:** Grand map image will be kept with this map image.

**Constructor**

Initial view of the grand map will be composed by constructor.

**Methods**

**void updateView:** According to player actions and some random actions new views will be updated by this method.

**void actionPerformed:** Button events will be handled under this method.

**Void paintComponent:** Image map will be drawn by this method.

#### 4.4.2.5. Civics Panel

This class will be extended with JPanel.



**Attributes**

**JLabel government:** This is a basic label having government string on.

**JLabel economy:** This is a basic label having economy string on.

**JLabel tyranny:** This is a basic label having tyranny string on.

**JRadioButton oligarchy:** This is a radio button of oligarchy.

**JRadioButton democracy:** This is a radio button of democracy.

**JRadioButton tradeEconomy:** This is a radio button of easy level.

**JradioButton agrarianism:** This is a radio button of agrarianism.

**JRadioButton guilds:** This is a radio button of guilds.

**JRadioButton militia:** This is a radio button of militia.

**JRadioButton mercenaries:** This is a radio button of mercenaries.

**JRadioButton standingArmy:** This is a radio button of standing army.

**JTextArea bonuses:** Bonuses of the selected government type will be written in this text area.

**JButton cancel:** Player can cancel changes with this button.

**JButton applyAndReturn:** Player can apply changes and return to city layer with this button.

**Constructor**

All buttons, radio button will be formed under constructor and default settings will be done too.

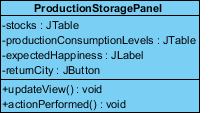
**Methods**

**void updateView:** According to player actions, text in the text areas will be updated with this method.

**void actionPerformed:** Button event will be done with this method.

#### 4.4.2.6. Production Storage Panel

This class will be extended with JPanel.



**Attributes**

**JTable stocks:** This table will keep the resources names, amounts in storage, expected consumptions and productions.

**JTable productionConsumptionLevels:** This table will keep manufactured resources.

**JLabel expectedHappiness:** Player will see expected happiness of proletariat and aristoi.

**JButton returnCity:** This button will enable player to go back to the city layer.

**Constructor**

Initial view of the panel will be composed under constructor. The attributes will be formed too.

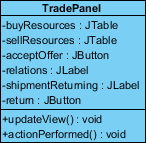
**Methods**

**void updateView:** According to player actions this method update the panel.

**void actionPerformed:** This method enable player to go back to city layer when player clicks on returnCity button.

#### 4.4.2.7. Trade Panel

This class will be extended with JPanel.



**Attributes**

**JTable buyResources:** This table will keep the resource which can be bought with their type, cost, amount etc.

**JTable sellResources:** This table will keep the resource which can be sold with their type, cost, amount etc.

**JButton acceptOffer:** Player can buy or sell resources by clicking this button.

**JLabel relations:** Relations with other countries will be showed with this label.

**JLabel shipmentReturning:** Ships with their load and returning time will be showed with this label.

**JButton return:** This button enable player to go back to the city layer.

**Constructor**

Initial view of the trade panel will be composed. Also attributes that defined above will be formed.

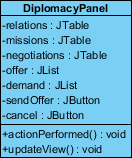
**Methods**

**void updateView:** According to player actions this method update the panel.

**void actionPerformed:** Button events will be handled.

#### 4.4.2.8. Diplomacy Panel

This class will be extended with JPanel.



**Attributes**

**JTable relations:** This table will keep relations with other countries.

**JTable missions:** Active missions will be kept in this table.

**JTable negotiations:** This table will keep negotiation with other countries.

**JList offer:** List of offers will be kept.

**JList demand:** List of demands will be kept.

**JButton sendOffer:** Player can send an offer via this button.

**JButton cancel:** Player can cancel negotiations with this button.

**Constructor**

Initial view will be composed and attributes will be formed.

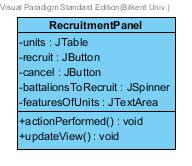
**Methods**

**void actionPerformed():** Button events will be done under this method.

**void updateView():** According to player action, panel will be updated.

#### 4.4.2.9. Recruitment Panel

This class will be extended with JPanel.



**Attributes**

**JTable units:** This table will keep units type and their requirement like sword, armor.

**JButton recruit:** Player can apply recruitment with this button.

**JButton cancel:** Player can cancel recruitment with this button.

**JSpinner battoliansToRecruit:** Player can increase and decrease amount of unit with this spinner.

**JTextArea featuresOfUnits:** In this area, features of units will be written.

**Constructor**

Initial view will be composed and attributes will be formed.

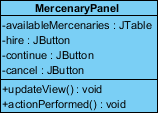
**Methods**

**void actionPerformed:** Button actions will be handled.

**void updateView:** According to player action, panel will be updated.

#### 4.4.2.10. Mercenary Panel

This class will be extended with JPanel.



**Attributes**

**JTable availableMercenaries:** This table will include some information about merecenaries like their leader, forces, cost etc.

**JButton hire:** This button enables to hire a mercenary army.

**JButton continue:** Player can continue to city layer by clicking this button.

**JButton cancel:** Player cancel hiring via this button.

**Constructor**

Initial view of mercenary panel will be composed by forming attributes above.

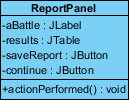
**Methods**

**void updateView:** This method will update the mercfenary screen according to player actions.

**void actionPerformed:** Button actions will be done under this method.

#### 4.4.2.11. Reports-Notification Panel

This class will be extended with JPanel.



**Attributes**

**JLabel aBattle:** Name of the battle will be written on this label.

**JTable results:** Results of the battle will be shown on this table.

**JButton saveReport:** Player can save the battle report with this button.

**JButton continue:** Player can continue to game by clicking this button.

**Constructor**

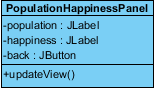
Attributes will be composed.

**Methods**

**void acitonPerformed:** Buttons actions will be done.

#### 4.4.2.12.Population Happiness Panel

This class will be extended with JPanel.



**Attributes**

**JLabel population:** Population of aristoi and proletariat will be shown on this label.

**JLabel happiness:** Happiness of aristoi and proletariat will be shown on this label.

**JButton back:** Player can go back to city layer with ths button.

**Constructor**

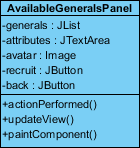
Initial view of the panel will be composed adding labels and the button above.

**Methods**

**void updateView:** While population and happiness are changing this method enables to update the panel.

#### 4.4.2.13.Available Generals Panel

This class will be extended with JPanel.



**Attributes**

**JList generals:** There will be a list that includes names of available generals.

**JTextArea attributes:** Attributes of generals will be explained in text areas.

**Image avatar:** Generals will have images next to their attributes.

**JButton recruit:** Player can recruit a general with this button.

**JButton back:** This button enables player to go back to the city layer.

**Constructor**

Initial view of the panel will be composed.

**Methods**

**void actionPerformed:** Button actions will be defined.

**void updateView:** According to available generals, list will be updated. Available generals can be occur according to some calculations and some random values.

**void paintComponent:** Images of generals will be drawn.

### 4.4.3. Sound Manager

This subsystem will have two classes: SoundManager class and SoundListener class. SoundListener class will be implemented inside SoundManager class as a private class. And SoundListener will implement ActionListener.



**Attributes**

**MediaPlayer sound:** This attribute will be used to benefit from the media package of Java.

**Constructor**

The attribute, sound, will be composed.

**Methods**

**void actionPerformed:** Methods of media packages will be used under actionPerformed method. These methods are play(), pause() and setMute(boolean value).

### 4.4.4.Game Element Classes

#### 4.4.4.1.Material

This class is an abstract class extended by RawMaterial and ManufacturedMaterial classes. Represents the resources in the game.

#### 

**Attributes:**

**amount:**Refers to the amount that player have currently, since every material will be instantiated only once.

**baseTradeCost:** Used when a trade offer is being formed.

**requiredAmountToFillShip**:How much of this resource can fill a ship, during trade shipments, again used when forming trade offers.

**allowedToConsume**: Some materials consuming can be restricted by the player, holds this choice.

#### 4.4.4.2.RawMaterial

Extends Material Class, represents the resources can be gathered from nature-provinces. Extended by SeasonalMaterial class.



**Attribute**:

**isGathererSlave**:refers to the gatherer type in Sicily Map, can be a slave or a proletariat.

#### 4.4.4.3.SeasonalMaterial

Extends RawMaterial Class, represents the agriculture products that can be harvested only in certain months.

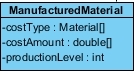


**Attribute**:

**season**:can change between 1-4 depending on the harvest season, 1 is winter.

#### 4.4.4.4.ManufacturedMaterial

Extends Material Class, represents the materials produced in the city.



**Attributes**:

**costType**:Material types that used during the production of this material.

**costAmount**:Material amounts corresponds to the costType, for example if 1 bread is made from 0.2 flour, costType[0] is bread while costAmount[0] is 0.2.

**productionLevel**:User can change the production level of each manufactured material, holds this value, can be change between 0 and 2 0 is not producing, 2 is maximum production, 1 is produce enough to meet domestic need.

#### 4.4.4.5.Sicilopedia

Represents the Sicilopedia of our game.



**Attribute**:

**rootCategory**:Since Sicilopedia is also a category of Sicilopedia categories that includes entries and other categories, Sicilopedia is basically a kind of special SicilopediaCategory.

#### 4.4.4.6.SicilopediaObject

SicilopediaObject is extended by SicilopediaCategory and SicilopediaEntry.



**Attribute**:

**title**:Every object in Sicilopedia, whether a category or entry, has a title.

#### 4.4.4.7.SicilopediaCategory

Extends SicilopediaObject, represents categories of the Sicilopedia.



**Attribute**:

**content**:Every SicilopediaCategory holds some SicilopediaObjects in it, like a folder holds in itself both folders and the files.

#### 4.4.4.8.SicilopediaEntry

Extends SicilopediaObject, represents entries of the Sicilopedia.



**Attribute**:

**quote**:Every Sicilopedia entry includes a related quote at the beginning.

strategy:Every Sicilopedia entry includes a game related knowledge about the subject.

**information**:Every sicilopedia entry includes general knowledge about the subject.

#### 4.4.4.9.People

Abstract class of different social classes in the Syracuse, extended by FreePeople and Slave classes.



Attributes:

**countTotal**:Number of this social class’s members in the whole game, controlled by the player.

**countInCounties**:Number of this social class’s members living in the countryside, provinces.

**countInArmy**:Number of this social class’s members that are currently in armed forces.

**countInCity**:Number of this social class’s members that are living in the city-Syracuse.

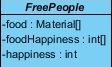
**unemployed**:Number of this social class’s members that are unemployed currently.

Method:

**consume**:Abstract method that is implemented by Slaves,Proletariat, and Aristoi. These classes takes the material array of the player, and consume proportional to their total population, if class is Proletariat or Aristoi, these classes update their happiness. If ifBesieged boolean is true, people living in the countryside are not calculated during consumation.

#### 4.4.4.10.FreePeople

Abstract class that Extends People class, extended by Proletariat and Aristoi.



**Attributes**:

**food**:Food materials that the instance of this object consumes during the consume method.

**foodHappiness**:For each material in the food array, corresponds to the happiness gained from this resource. Can be change between 1 and 3.

happiness:Current happiness of the people, between 0 and 100.

#### 4.4.4.11.Slave

Class that extends People class. Does nothing except a unique implementation of consume method.



#### 4.4.4.12.Proletariat

Class that extends FreePeople class. Implements the abstract consume method.



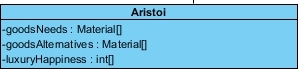
**Attributes**:

**goodsNeeds**: Represents the non-food needs of Proletariat people.

**goodsHappiness**:When met, represents the happiness gained from materials in the goodsNeeds array.

#### 4.4.4.13.Aristoi

Class that extends FreePeople class. Implements the abstract consume method.



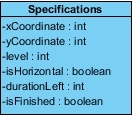
**goodsNeeds**: Represents the non-food needs of Proletariat people.

**luxuryHappiness**:When met, represents the happiness gained from materials in the goodsNeeds array.

**goodsAlternatives**:When the needs in the goodsNeeds not met, Aristoi people consume the resources in this array, like if there’s dyed clothes in the goodsNeeds but there’s no dyed clothes in the player, consume method checks the same index of the dyed cloth of the goodsNeeds in the goodsAlternatives array, and consumes from this resources. However consumation of these resources does not give any happiness.

#### 4.4.4.14.Specifications

Represents the values that make every instance of a building unique. Like the barcode of a book’s instances.



**Attributes**:

**xCoordinate**:Top-left X coordinate of the building in the city map.

**yCoordinate**:Top-left Y coordinate of the building in the city map.

level:Upgrade level of the instance.

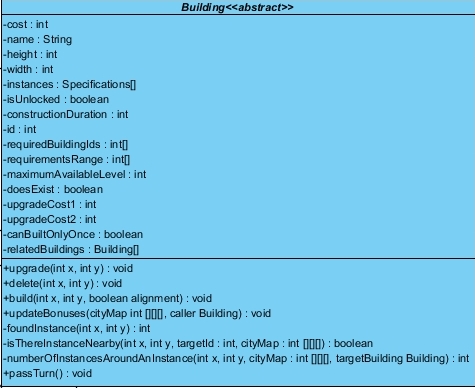
**isHorizontal**:Whether building places horizontally or vertically, if vertical width used as Y length of the building etc.

**durationLeft**: If building is in construction, how many months left to finish.

**isFinished**: Whether the building is already constructed or still being constructed.

#### 4.4.4.15. Building

The base and abstract Building class of the game, every building is somewhat descendant of this class.



**Attributes**:

**cost**:Money cost of the building’s construction.

**height-width**:Height and width of an instance of this building.

instances:holds the instances of this building.

**id**:Id used in city map for fast calculation of building distances.

**requiredbuildings**:Array of required buildings to be checked before building an instance.

**requirementsRange**:Each range corresponds to a required building during range check.

**maximumAvailableLevel**:The level that limits the level of the instances of this building.

**upgradeCost1-2**:Money cost during the upgrades to the level 2 and 3 respectively.

**relatedBuildings**:updatebonuses methods of these buildings will be called, whenever an instance of this building is built, deleted or upgraded.

**Methods**:

**upgrade**:Upgrades the instance of the building that occupies the given x and y values.

**delete**:deletes the instance of the building that occupies the given x and y values.

**build**:Builds an instance of this building.

**updateBonuses**: Called when an instance of this building is built, deleted, or upgraded. Also can be called from the relatedBuildings part of related buildings. Base implementation has empty body, but every range related building implements its own.

**foundInstance**:Returns the “specifications” index of given coordinates.

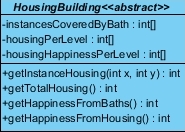
**isThereInstanceNearby**:Check whether there is an id in the city map array close to the x and y coordinates.

**numberOfInstancesAroundAnInstance**:Returns the number of instances of the given building type within the range of the given building instance with x and y coordinates.

**passTurn**:Reduces left construction duration of all instances that are being built.

#### 4.4.4.16.Housebuilding

Extends Building class, parent class for housing related buildings.



**Attributes**:

instancesCoveredByBath:Since every housing building can be covered only once by a bath, holds the indexes of “specifications” to prevent duplicates.

**housingPerLevel**:Array that has three values for three levels of housing capacities of instances.

**housingHappinessPerLevel**:Array that has three values for three levels of housing happiness of instances.

**Methods**:

**getInstanceHousing**:Returns the housing capacity of a chosen instance.

**getTotalHousing**:Returns the total housing provided by all instances of this building.

**getHappinessFromBaths**:Returns the total happiness gained from public baths.

**getHappinessFromHousing**:Returns the total happiness gained from this type of object.

#### 4.4.4.17.House

Extends HousingBuilding class, represents the houses of proletariat.



**Attribute**:

**housesCoveredByShrines**:Since every house can be covered by at most one shrine, holds the all instances of covered houses to prevent duplicates, also holds the covering shrine’s level since it provides different happiness.

**Method**:

**getHappinessFromShrines**:Returns total happiness gained by shrine coverage over the houses.

#### 4.4.4.18.Mansion

Extends HousingBuilding class, represents the houses of Aristoi.



**Attribute**:

**mansionsCoveredByTemples**:Since every manison can be covered by at most one temple, holds the all instances of covered mansions to prevent duplicates.

**Method**:

**getHappinessFromTemples**:Returns total happiness gained by temple coverage over the mansions.

#### 4.4.4.19.Road

Extends Building class. Nothing special. Handled by manager classes.



#### 4.4.4.20.Agora

Extends Building class. Nothing special. Handled by manager classes.



#### 4.4.4.21.WorkBuilding

Abstract class that represents the buildings that requires workers in them. Extends Building class.



All extenders implement their own required people methods. These methods return the workers working on an instance, so when an instance created or deleted can be used by population managers.

#### 4.4.4.22.ArcheryRange

Extends WorkBuilding class. Nothing special. Handled by manager classes.



#### 4.4.4.23.SiegeWorkshop

Extends WorkBuilding class. Nothing special. Handled by manager classes.



#### 4.4.4.24.Library

Extends WorkBuilding class. Nothing special. Handled by manager classes.



#### 4.4.4.25.PublicBath

Extends WorkBuilding class.



**Attribute**:

**range**:Represents the range of the baths during happiness calculations

**Method**:

**updateBonuses**:Overrides the Building class method, updates the instances of covered mansions and houses.

#### 4.4.4.26.GeneralsHeadquarter

Extends WorkBuilding class. Nothing special. Handled by manager classes.



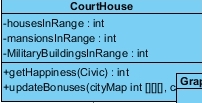
#### 4.4.4.27.Barrack

Extends WorkBuilding class. Nothing special. Handled by manager classes.



#### 4.4.4.28.Courthouse

Extends WorkBuilding class.



**Attributes**:

**housesInRange**:Holds the number of house instances covered by courthouse.

**mansionsInRange**:Holds the number of house instances covered by courthouse.

**MilitarybuildingsInRange**:Holds the number of house instances covered by courthouse.

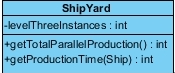
**Methods**:

**getHappiness**:Manager class sends the current government civic, and according to this parameter, courthouse returns a happiness value depending on its attributes.

**updateBonuses**:Override of the Building Class Method, courthouse updates its attributes.

#### 4.4.4.29.Shipyard

Extends WorkBuilding class.



**Attribute**:

**levelThreeInstances**:Provides direct access to the number of last level instances.

Methods:

**getTotalParallelProduciton**:Returns the number of ships that can be built simultaneously.

**getProducitonTime**:Returns the number of turns takes to construct the given ship as a parameter, if number of current ships that are in production is not exceeding **levelThreeInstances**, returns a reduced number.

#### 4.4.4.30.LightHouse

Extends WorkBuilding class. Handled by manager classes.



#### 4.4.4.31.Palace

Extends WorkBuilding class.



**Method**:

**getMaximumAnarchy**:When asked by the city manager during civic changes, returns the number of turns that must pass under anarchy according to the level of the palace.

#### 4.4.4.32.Theatre

Extends WorkBuilding class.



**Attributes**:

**range**:The range of the theatre for bonus calculations

**numberOfHousesInRange**:The number of houses covered by the range of the theatre.

**numberOfMansionsInRange**:The number of mansions covered by the range of the theatre.

**isUsingDye**:Increases returned happiness if using.

Methods:

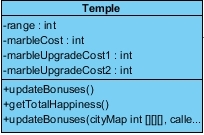
**getProletariatHappiness**:Returns proletariat happiness proportional to the houses in range.

**getAristoiHappiness**:Returns aristoi happiness proportional to the mansions in range.

**updateBonuses**:Overrides the Building class method, updates the attributes of the theatre.

#### 4.4.4.33.Temple

Extends WorkBuilding class.



**Attributes:**

**Range:**The range of the temple for happiness coverage calculations.

**marbleCost:** Marble cost for building a temple.

**marbleUpgradeCost1:** Marble cost for upgrading an instance to level 2.

**marbleUpgradeCost2:** Marble cost for upgrading an instance to level 3.

**Methods:**

**updateBonuses:** One without parameters shouldn’t be there but seems to be forgotten, ignore.

**getTotalHappiness:** Returns the happiness from monuments and shrines in the range of temples.

**updateBonuses:** Updates the mansions covered by temples, and monument and shrine numbers covered by temples.

#### 4.4.4.34.Shrine

Extends WorkBuilding class.



**Attribute:**

**range:** Base range of a Shrine for bonus coverage calculations.

**Method:**

**updateBonuses:** Updates the house instances covered by shrines.

#### 4.4.4.35.Academy



**Attribute**

**isObservatoryNearby:** Attribute for holding whether academy is near a observatory.

**Method**

**updateBonuses:** Checks whether a observatory nearby.

#### 4.4.4.36.Observatory

Extends WorkBuilding class. Handled by manager classes



#### 4.4.4.37.SlaveTrader

Extends WorkBuilding class.

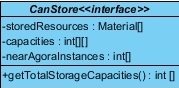


**Method**

**getSlaveTradingModifier:**Depending on the number of instances returns a slave trading modifier to be used during trade offer calculations.

#### 4.4.4.38.CanStore

An interface to achieve multiple inheritance in the Game.



**Attributes**

**storedResources:** Type of materials that are stored by implementer class.

**capacities:** For each type in storedResources array, corresponds to the capacity of the storage by each instance of the building.

**nearAgoraInstance:** Attribute used during calculations.

**Method**

**getTotalStorageCapacities:** For each storedResources returns the maximum capacity that this resource can be stored in the city.

#### 4.4.4.39.MetalMerchant

Implements CanStore interface and extends Workbuilding class.



#### 4.4.4.40.GrapeJuiceMerchant

Implements CanStore interface and extends Workbuilding class.



#### 4.4.4.41.Granary

Implements CanStore interface and extends Workbuilding class.



#### 4.4.4.42.OilMerchant

Implements CanStore interface and extends Workbuilding class.



#### 4.4.4.43.SaltMerchant

Implements CanStore interface and extends Workbuilding class.



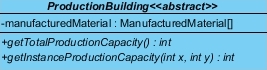
#### 4.4.4.44.CheeseMerchant

Implements CanStore interface and extends Workbuilding class.



#### 4.4.4.45.ProductionBuilding

Extends WorkBuilding class, represents the buildings that produce manufactured materials.



**Attribute**

**manufacturedMaterial:** Holds which type of materials are produced by this building.

**Methods**

**getTotalProductionCapacity:** Returns the total production capacity for the materials above.

**getInstanceProductionCapacity:** Returns the production capacity of a single instance.

#### 4.4.4.46.OilPresser

Extends ProductionBuilding class.



#### 4.4.4.47.Bakery

Extends ProductionBuilding class.



**Method**

**getTotalFlourUse:** Since upgraded bakeries reduce the flour consumption, manager classes ask this method to return total flour use.

#### 4.4.4.48.GrapeJuiceDistillery

Extends ProductionBuilding class.



#### 4.4.4.49.ProductionAndStorageBuilding

Extends ProductionBuilding class, and implements CanStore interface.



#### 4.4.4.50.WeaponMaker

Extends ProductionAndStorageBuilding.



**Attribute**

**ArmoriesAndWeaponMakersNearby:**Holds the number of nearby armories and eaponries for bonus calculations, for each instance.

**Method**

**updateBonuses:** Updates nearby bonuses.

#### 4.4.4.51.FabricMerchant

Extends ProductionAndStorageBuilding.

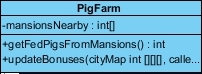


**Method**

**getTotalDyeUse:** Since level of fabric merchant affects its dye use, manager classes acceses total dye consumption from this building.

#### 4.4.4.52.PigFarm

Extends ProductionAndStorageBuilding.



**Attribute**

**mansionsNearby:** Holds the number of mansions nearby for every instance of PigFarm.

**Method**

**getFedPigsFromMansions:** Returns the number of pigs produced without using barley.

**updateBonuses:** Updates mansion bonuses.

#### 4.4.4.53.Armory

Extends ProductionAndStorageBuilding.



**Attribute**

**ArmoriesAndWeaponMakersNearby:**Holds the number of nearby armories and eaponries for bonus calculations, for each instance.

**Method**

**updateBonuses:** Updates nearby bonuses.

#### 4.4.4.54.LumberMerchant

Extends ProductionAndStorageBuilding.



#### 4.4.4.55.Butchery

Extends ProductionAndStorageBuilding.



#### 4.4.4.56.GlassMaker

Extends ProductionAndStorageBuilding.



#### 4.4.4.57.Fishery

Extends ProductionAndStorageBuilding.



#### 4.4.4.58.Mill

Extends ProductionAndStorageBuilding.



#### 4.4.4.59.MarbleCutter

Extends ProductionAndStorageBuilding.



#### 4.4.4.60.PotteryMaker

Extends ProductionAndStorageBuilding.



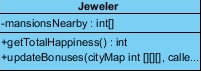
#### 4.4.4.61.Stable

Extends ProductionAndStorageBuilding.



#### 4.4.4.62.Jeweler

Extends ProductionAndStorageBuilding.



**Attribute**

**mansionsNearby:** For every instance of Jeweler, the number of nearby mansions.

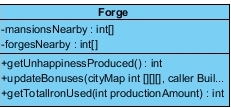
**Methods**

**getTotalHappiness:** Total happiness gained from these mansions.

**updateBonuses:** Update mansionsNearby.

#### 4.4.4.63.Forge

Extends ProductionAndStorageBuilding.



**Attributes**

**mansionsNearby:** For every instance of a forge, the number of nearby mansions.

**forgesNearby:** For every instance of a forge, the number of nearby forges.

**Methods**

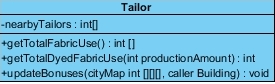
**getUnhappiness:** Returns the unhappiness caused by forges nearby mansions.

**updateBonuses:** Updates both attributes.

**getTotalIronUsed:** Since Iron ore used is affected by forge bonuses, managers call this method to get used iron ore.

#### 4.4.4.64.Tailor

Extends ProductionAndStorageBuilding.



**Attribute**

**nearbyTailors:** For every instance of a tailor number of tailors nearby.

**Methods**

**getTotalFabricUse:** Returns the fabric used by tailors, since it’s affected by tailor bonuses.

**getTotalDyedFabricUse:** Returns the dyed fabric used by tailors, since it’s affected by tailor bonuses.

**updateBonuses:** Update nearby taior bonuses.

#### 4.4.4.65.TradeOffer

Represents the trade offers offered to the player.



**Attributes**

**sender:** Sender faction.

**resource:** Material type to be traded.

**amount:**Material amount to be traded

**isFactionExporting:**True if the faction is selling.

**cost:**Money cost of the material.

**isFullShipment:**Whether this amount of resource fills the ship or half of it.

**duration:**Duration left for approving the offer.

#### 4.4.4.66.TradeShipment

Represents the trade shipments currently on the way.



**Attributes**

**exportedMaterial:**Material types that are being exported.

**importedMaterial:**Material types that are being imported.

**exportAmount:**Amount of materials that are being exported.

**importAmount:**Amount of materials that are being imported.

**totalMoneyChange:**After the shipment returns, the money change in player’s treasure.

**monthsLeft:**how many turns left for the shipment to return to Syracuse.

**longDistanceShipment:**Whether shipment goes to a 2 month long distance or 1. True if trading with Seleucids for example.

#### 4.4.4.67.Faction

Base class for all faction types in the game.



**Attributes**

**name:**Name of the faction.

#### 4.4.4.68.PlayerFaction

Extends Faction. Actually does not do anything, since provinces armies etc. already held in game object. But required for province-army pointers.

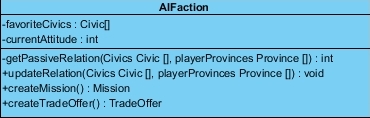


**Attribute**

**provinces:** Provinces held by player.

#### 4.4.4.69.AIFaction

Extends Faction class, base class for all non-player factions. Example object is first age Corinth.



**Attributes**

**favoriteCivics:**An array of favorite civics, compared to user’s civics during passive relation calculations.

**currentAttitude:**Current relation points toward to player.

**Methods**

**getPassiveRelation:**Compares civics and raw materials produced by player to calculate a passive relation point.Overridden by subclasses, but we forgot to show on diagrams.

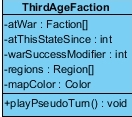
**updateRelation:**Merges currentAttitude with passive relation.

**createMission:**Returns a mission object to player to accomplish. Overridden by subclasses, but we forgot to show on diagrams.

**createTradeOffer:**Creates trade offer depending on faction’s resources-regions, relation etc. Overridden by subclasses, but we forgot to show on diagrams.

#### 4.4.4.70.ThirdAgeFaction

Extends AIFaction class, represents factions of the third age.



**Attributes**

**atWar:** Factions that this faction is in war with.

**atThisStateSince:** If the faction is in a war, since how long this battle continues, or if factions is in peace since when. As number increases faction tends to change its state, like declare war to a faciton if it’s in peace for 20 years.

**warSuccesModifier:** Faction’s success rate while conquering regions on pseudo turns.

**Regions:**Regions held by this faction.

**mapColor:**Color that represents this faction’s regionsç

**Method**

**playPseudoTum:**Faction play turns, randomly declares war-peace, creates trade offers, missions, conquers regions etc.

#### 4.4.4.71.SicilyFaction

Extends ThirdAgeFaction. Represents the Carthage, Rome and rebel forces on Sicily.



**Attributes**

**provinces:**Provinces held on Sicily.

**armies:**Armies in Sicily.

**navies:**Navies in Sicily.

**lastExpeditionSince:**How many months passed since this faction sent forces to Sicily, higher value means stronger expedition armies will be sent, since the faction is revived.

**flag:**Flag of the faction to use on sicily and diplomacy map.

**averageArmyComposition:**On average, armies sent by this faction to the Sicily, when an expedition occurs the sent army will be based on this army and only will change in numbers depending on lastExpeditionSince.

**averageNavyComposition:**Same thing for navies.

**capital:**Capital region of the faction, if lost and user is ally of the conqueror, user wins the game.

**atWarWithPlayer:**Whether the faction is in war with player.

**isRebel:**To differentiate whether faction is a faction of mercenaries and rebels, bad object oriented practice, but we didn’t had enough time to think through it.

**tradeExceptionToThePlayerLeft:**The turns left for the trade exception given by faction.

**tradeExceptionFromPlayerLeft:**The turns left for the trade exception given by player.

**tradePromiseLeft:**The turns left for this faction that forbids it with trading others.

**Methods:**

**createArmy:**Create an expedition army on Sicily, based on averageArmycomposition and lastExpeditionSince values.

**createNavy:**Same thing for navies.

**playRealTurn:**If faction is at war in Sicily calls this method to move his armies, conquer provinces etc.

**createDiplomaticOffer:**Creates a diplomatic offer to be sent to the player, based on current relations.

**gradeDiplomaticOffer:**Grades an offer sent by the player and agrees or not.

#### 4.4.4.72.PeloponnesianFaction

Extends AIFaction class, Athenians and Spartans are instances.



**Attributes**

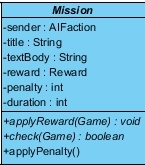
**producedResources:**Resources that this faction offers on trade.

**expeditionArmy:**When the deadline given at the end of 2nd age finishes, this army will sent to the Player.

**expeditionNavy:**Same thing for navy.

#### 4.4.4.73.Mission

The abstract base mission class. All methods are implemented by children, except the penalty.



**Attributes**

**sender:**Sender Faction.

**Title:**Title of the mission.

**textBody:**Description of the mission.

**Reward:**Reward that is attached to the mission, applied when mission check returns true.

**penalty:**Relationship penalty, if the mission fails.

**duration:**Mission deadline turns left.

**Methods**

**applyReward:**Applies reward on the game object, has access to all game object.

**check:**Checks whether mission is accomplished or not.

**applyPenalty:**Reduces penalty from the relationship with the faction.

#### 4.4.4.74.DeclareWar

Mission type given by Peloponnesian Factions to declare war on another Peloponnesian Faction. Check is made from diplomatic relations.



**Attribute**

**faction:** Faction to be declared war.

#### 4.4.4.75.BuildBuilding

Mission that wants player to build a specific building instance.



**Attribute**

**building:** Building to be constructed.

#### 4.4.4.76.StopTrading

Mission that wants player to stop trade with another faction.



**Attribute**

**faction:** Faction to stop trade.

#### 4.4.4.77.RecruitSoldier

Mission type that wants player to recruit certain kind of soldier.



**Attribute**

**unit:** Soldier type to be recruited.

#### 4.4.4.78.CivicChange

Mission that is given by other factions to the player to change his civics.



**Attribute**

**civic:**Civic to be adopted.

#### 4.4.4.79.ResourceDemand

Mission that forces player to within certain months accepting a trade offer that demands resources.



**Attributes**

**type:**Material type.

**amount:** Amount of material.

#### 4.4.4.80.SendAid

Sicily factions sends this mission if they are player’s ally and their city is besieged, player must lift the siege.



**Attribute**

**destination:** City Province that is besieged and must be rescued.

#### 4.4.4.81.Reward

Reward class that is a base for all other reward classes.



**Attribute**

**relationBonus:**Every accomplished mission increases relation in someway, so this is the amount of attitude to gain from the AI.

**Method**

**Apply:** All subclasses of Reward, overrides this method, but we forgot to include them in diagram. Method applies the reward on the game object.

#### 4.4.4.82.DeadlineReward

Rewards that extends the deadline in the first age.



**Attribute**

**months:** Months to be extended.

#### 4.4.4.83.SoldierReward

Reward that gifts soldiers to the player.



**Attribute**

**unit:** Type of unit to be gifted.

#### 4.4.4.84.ResourceReward

Reward type that increases materials of the player.



**Attributes**

**type:**Type of resource that is given.

**amount:** Amount.

#### 4.4.4.85.DiplomaticOffer

DiplomaticOffer objects holds the information to conduct diplomatic relations.



**Attributes**

**side:**Sicily faction that is negotiating with player.

**moneyPlayerGains:**Money given to player. Can be negative.

**moneyplayerReceivesPerMonth:**Money player regularly gains for “ReceivingMonths”.

**ReceivingMonths:** Number of months that regular payment is conducted.

**moneyPlayerGivesPerMonth:**Money player regularly pays for “givingMonths”.

**givingMonths:** Number of months that regular payment is conducted.

**tradeExceptionFromPlayer:**Number of months to be a trade exception to hold from player.

**tradeExceptionToPlayer:** Number of months to be a trade exception to hold from “side”.

**cancelTradePlayer:**Number of months that player is forbidden from trading other Sicily Factions.

**cancelTradeAI:**Number of months that AI is forbidden from trading other Sicily Factions.

**declareWarPlayer:** Player declares war to the other Sicily Faction.

**declareWarAI:**AI declares war to the other Sicily Faction.

**senAidToPlayer:**AI sends help to the player.

**sendAidToAI:**Player has to send help to the AI.

**adaptGovernmentalCivic:**Player adapts Faction’s favorite Governmental Civic.

**adaptEconomicCivic:**Player adapts Faction’s favorite EnomicCivic.

**adaptMilitaryCivic:**Player adapts Faction’s favorite MilitaryCivic.

**Methods**

**approve:**Offer is conducted, and applied on game.

**getEstimateResult:**Estimate result during the diplomacy screen, whether AI tends to say yes or no.

#### 4.4.4.86.Place

Base class for Provinces and Regions on the game.



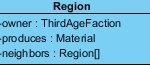
**Attributes**

**name:**Name of the place.

**pixelsOnTheMap:**For regions, pixels on Grand Map image to be painted, for Provinces Sicily Map.

#### 4.4.4.87.Region

Extends Place class, represents the places on the Grand Map.



**Attributes**

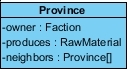
**Owner:** Owner is a third age faction.

**Produces:** Produces a tradable material.

**neighbors:** Neighbor regions’ pointers for conquering-territorial purposes.

#### 4.4.4.88.Province

Extends Place class, represents places on the Sicily map.



**Attributes**

**owner:**Owner can be player too, so no AIFaction limit.

**produces:**This was actually removed to be included in “County” but visual paradigm crashed without saving, and now I have no time to fix and replace the image, ignore.

**neighbors:** Other neighbor provinces in the Sicily map.

#### 4.4.4.89.SeaProvince

Represents sea provinces in the Sicily map, extends Province class.



**Attribute**

**Navy:** A sea province might include a navy.

#### 4.4.4.90.LandProvince

Represents land provinces in the Sicily map, extends Province class.

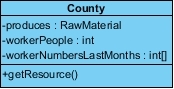
#### 

**Attribute**

**army:** A land province might include an army.

#### 4.4.4.91.County

Extends LandProvince class, represents the land provinces that are not cities.



**Attributes**

**produces:** Every County produces a raw material type.

**workerPeople:**Number of working people for the player in this region.

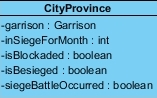
**workerNumberLastMonth:**To prevent user from abusing the seasonal harvest mechanic, worker numbers of previous months are also stored, so player must also send workers during non-harvest seasons too.

**Method**

**getResources:** This is supposed to return int, but I forgot to include in diagram, returns the raw material amount that is gathered from the province.

#### 4.4.4.92.CityProvince

The CityProvince class extends the LandProvince class. Represents the city tiles in Sicily.



**Attributes**

**garrison:**Garrison forces in the city.

**inSiegeForMonth:**The number of months that city is in siege.

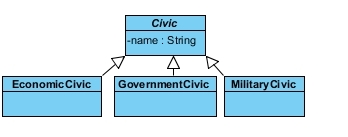
**isBlockaded:**Is ports of the city blockaded.

**isBesieged:**Is city is currently being besieged.

**siegeBattleOccured:**Did ever a siege battle occurred in this siege.

#### 4.4.4.93.Civic

Civic classes and its descendants are not straightforward classes to be implemented as objects, so they are mainly handled by manager classes.

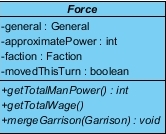


**Attribute**

**Name:** Name of the civic.

#### 4.4.4.94.Force

Force is an abstract class that forms the basis of all military groups in the Sicily.



**Attributes**

**general:** A force might have a general.

**approximatePower:** Every force has an approximate power, not used for calculating actual battles, but for rebellion purposes.

**faction:**Faction that controls this force.

**movedThisTurn:**Every force can move once every turn.

**Methods**

**getTotalMenPower:**Abstract function that returns total man power in the force, implemented different in each subclass. But not shown in diagrams.

**getTotalWage:**Abstract function that returns total wage of the force. Implemented different in each subclass. But not shown in diagrams.

**mergeGarrison:** Handles the merging of a force to an army.

#### 4.4.4.95.Garrison

Garrison class extends Force class. Garrison represents the force that is in a city.



**Attributes**

**ships:**Array of ship types in the city.

**shipCount:**Array corresponds to the count of each type in the ships array.

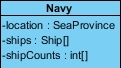
**units:**Array of land unit types in the city.

**unitCount:**Array corresponds to the count of each type in the units array.

**city:** The city that garrison stays.

#### 4.4.4.96.Navy

Extends Force class.



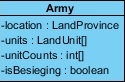
**Attributes**

**location:** Sea province that navy is patrolling.

**ships:**Array of ship types in the navy.

**shipCount:**Array corresponds to the count of each type in the ships array.

#### 4.4.4.97.Army



**Attributes**

**location:**Land province that army is currently in.

**units:**Array of land unit types in the army.

**unitCounts:**Array corresponds to the count of each type in the units array.

**isBesieging:**If army is besieging a city, its wages increases.

#### 4.4.4.98.MercenaryArmy

Extends Army class.



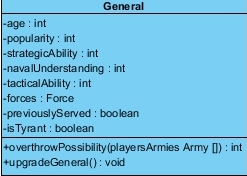
**Attributes**

**initialCost:**Initial cost of hiring the mercenary group.

**wage:**This attribute is no longer needed since every subclass implements its own wage getter method, but it’s forgotten here, ignore.

#### 4.4.4.99.General

Represents generals in the game.



**Attributes**

**age:**Age of the general, generals can die because of old age.

**popularity:**Represents the popularity of the general, increases happiness from victories. Too popular generals might rebel.

**strategicAbility:**Strategic ability of the general, important when commanding large forces.

**navalUnderstanding:**Naval ability of the general, important in sea battles.

**tacticalAbility:**Tactical ability of the general, important when commanding small forces.

**forces:** Force that general is currently controlling, if any.

**previouslyServed:** Previously served generals stay as available generals in the pool.

**isTyrant:**Tyrants produce more happiness from victories, and cannot rebel.

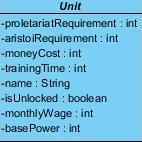
**Methods**

**overthrow Possibility:**Calculates and returns overthrow possibility of the general, popularity and large force compared to rest of the player’s armies increases chances.

**upgradeGeneral:**General is randomly upgraded in one of his attributes, popular generals has chance to gain more increase.

#### 4.4.4.100.Unit

Abstract class that represents the units in the game.



**Attributes**

**proletariaRequirement:**Required for one instance of the unit.

**aristoiRequirement:**Required for one instance of the unit.

**moneyCost:** Required for one instance of the unit.

**trainingTime:** Required time for training one instance of the unit.

**name:**Name of the unit.

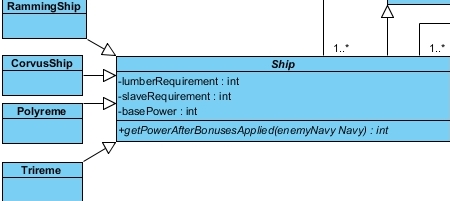
**isUnlocked:**Whether unit is locked to the player or not.

**monthlyWage:**monthly cost of one instance of unit.

**basePower:**Base power value for battle calculations.

#### 4.4.4.101.Ship

Ship class extends the Unit class, and has 4 children, each is a unique ship type.



**Attributes**

**lumberRequirement:**Lumber requirement for one instance of the ship.

**slaveRequirement:**Slave requirement for one instance of the ship.

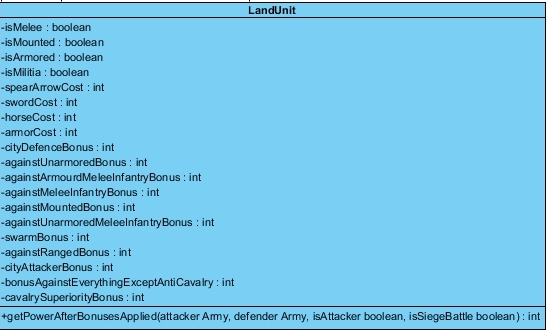
**basePower:** Forgot to remove, already inherited from base class, ignore.

**Method**

**getPowerAfterBonusesApplied:**Each of the children implements this method on its own and returns her power after bonuses applied.

#### 4.4.4.102.LandUnit

Extends Unit class, Each unique unit in the game is an object of this class. And every instance of a unit type points to these objects.



**Attributes:**

**Cost Attributes:**Straightforward costs for producing one instance of a unit object.

**Type Attributes:**Stored for applying bonuses against this object during fights, isMelee, is Mounted etc.

**Bonus Attributes:** Percentage power bonuses for each type of enemy, and for some special cases.

**Method**

**getPowerAfterBonusesApplied:**This method is called on an object of land unit, and this method looks at the enemy army and its units, and depending on these units and the unit that method is called, returns a calculated power to be used in battle calculations.

#### 4.4.4.103.Monument

Extends building class



**Attributes**:

**marbleCost**:marbleCost of a monument.

**associatedCivics**:For each monument active government civic during the time of constructions is recorded.

**Methods**:

**getHappinessForCivic**:Happiness granted by monuments for the parameter civic is returned.

#### 4.4.4.104.ClawOfArchimedes

Extends building class, handled by manager objects.



#### 4.4.4.105.Wonder

Extends Building class.



#### 4.4.4.106.GreatLightHouse

Extends Wonder class



#### 4.4.4.107.Colossus

Extends Wonder class



#### 4.4.4.108.StatueOfZeus

Extends Wonder class



#### 4.4.4.109.Parthenon

Extends Wonder class



### 4.4.5. Current Game Memory

#### 4.4.5.1.Player

This class’ objects hold the information related to player’s settings chosen during new game.



**Attributes**:

**playerEconomicSetting**:3 is for abundant 1 is for scarce, holds the economic difficulty.

**playerMilitarySetting**:3 is for easiest, 1 is for hard, holds the military difficulty.

#### 4.4.5.2.CityStatus

Holds some small scale knowledge to help game class.



**Attributes**:

**anarchyLeft**:Number of anarchy turns left in the city.

**isThereAFestival**:Whether city is in festival this turn.

**lastFestivalSince**:Number of months passed since last festival, new festival cannot occur if lower than 12.

#### 4.4.5.3.Game

Holds all the data related to the current session of the game. Probably going to be slightly modified during implementation.



**Attributes**:

**player**:Player object stored for keeping settings.

**people**:Array that holds 3 objects of people, slaves, aristoi, proletariat.

**buildings**:Holds one object of every building class, building instances are stored in specifications arrays.

**age**:Player’s current age in the game.

\***pol**:Civics of the player.

**date**:Historical date in months, start date is not decided.

**aiFactions**:Other factions that are currently in the game.

**activeMissions**:Active missions that player can accomplish.

**activeTradeOffers**:Active trade offers that player can accept

**activeShipments**:Active shipments that are currently in the sea.  
**resources**:One object for every material type is hold in this array.

**provinces**:Provinces owned by player

**armies**:Armies owned by player

**navies**:Navies owned by player

**garrisons**:Garrisons owned by player

**firstAgeDeadlineLeft**:If player is still in the first age, deadline for him to build a temple.

**expeditionDeadline**:If player is in the second age, deadline for invader army to come.

**availableMercenaries**:Available mercenary armies can be hired from mercenary window.

**availableGenerals**:Available generals can be taken from generals window..

**happinessFromWar**:Happiness gained from recent wars, decreases over time.

**units**:Special to every session because holds the unblocked values. So unit objects are also stored.

**cityStatus**:self-explanatory.

**money**:Total money of the player.

**historicalEvents**:Holds the changes happened in Grand Map.

**activeDiplomaticOffer**:If an offer is sent to the player, stored in this attribute, there can be at most one offer at the same time.

**trainingUnits**:Type of units that are currently being trained.

**leftTrainingMonths**:Months left for units in the trainingUnits array.

**cityMap**:A map of the city where each x,y coordinate is signed with the buildings’ id. Used for fast range calculations.

defeatedShipCount:Used for unblocking colossus wonder when the count is met.

### 4.4.6. Game Manager

#### 4.4.6.1.Manager

All other manager classes extend from this class.

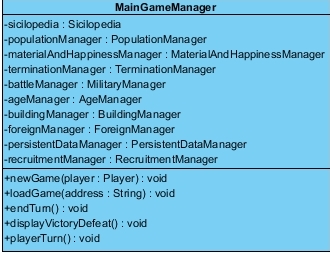


**Attribute**

**game:** Game object that every manager class has access.

4.4.6.2.MainGameManager

Deals with delegation of user inputs to other manager classes, and more general methods, extends manager class.



**Attributes:**

Attributes are self-explanatory.

**Methods**

**newGame:**Creates a new game object in itself, just from the user settings.

**loadGame:**Loads a game from an address.

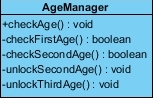
**endTurn:**When player ends turn, make function delegations inside this method.

**displayVictoryDefeat:**If the game is over, displays results.

**playerTurn:**Loop that player plays his turn.

#### 4.4.6.3.AgeManager

Extends Manager class.



**Methods**

**checkAge:**Checks whether player’s age is need to be changed, changes if true.

**checkFirstAge:**Check whether user passed the first age.

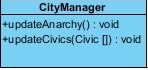
**checkSecondAge:**Check whether user passed the second age.

**unlockSecondAge:**Unlocks new things.

**unlockThirdAge:**Unlocks new things.

#### 4.4.6.4.CityManager

Extends manager class.

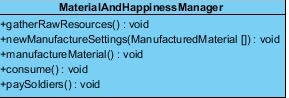


**Methods**

**updateAnarchy:**Reduces anarchy each turn, if there’s any. Creates anarchy if civics are changed.

**updateCivics:**Changes the civics of the player.

#### 4.4.6.5.MaterialAndHappinessManager



**Methods**

**gatherRawResources:**Deals with collection of raw resources to the player’s treasures.

**newManufacturedSettings:**Updates the manufactured materials’ production and consumption levels.

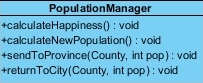
**manufacturedMaterial:**Deals with collection of manufactured materials.

**consume:**Deals with people’s consumption of the resources, and generation of happiness from them.

**playSoldiers:**Pays soldiers wages, reduces from treasury.

#### 4.4.6.6.PopulationManager

Extends Manager Class.



**Methods**

**calculateHappiness:**Calculates new turn’s happiness by looking at building happiness productions, wars and consumption etc.

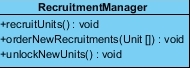
**calculateNewPopulation:**Calculates population changes.

**sendToProvince:**Sends chosen amount of people to the county from the city.

**returnToCity:**Sends chosen amount of people to the city from the country.

#### 4.4.6.7.RecruitManager

Extends Manager class.



**Methods**

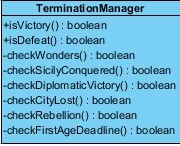
**recruitUnits:**Handles the recruitments of already ordered units when player ends turn.

**orderNewRecruitments:**During player’s turn, handles the creation of new orders.

**unlockNewUnits:**Unlocks new units when player ends turn, depending on some constraints.

#### 4.4.6.8.TerminationManager

Extends Manager class.



**Methods**

**isVictory:**Checks whether player won the game.

**isDefeat:**Checks whether player lost the game.

**checkWonders:**Checks whether player controls wonders.

**checkSicilyConqured:**Checks whether player conquered three cities of Sicily.

**checkDiplomaticVictory:**Checks whether player won a diplomatic victory.

**checkCityLost:**Checks whether player lost his city.

**checkRebellion:**Checks whether people rebel and overthrow the player.

**checkFirstAgeDeadline:**Checks whether first age deadline is met.

#### 4.4.6.9.ForeignManager

Extends Manager class.



**Methods**

**createShipment:**Creates a shipment object from the chosen trade offers by the player.

**moveShipment:**When turn ends, handles shipments movements.

**checkMissions:**Checks the missions, give rewards if any.

**AIFactionPlayTurn:**AI factions play their turns.

**approveOffer:**Applies diplomatic offer conditions, if user approves.

**createOffer:**Allows user to create diplomatic offers.

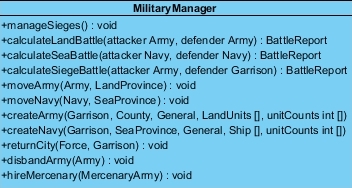
**responseToOffer:**Handles offers sent to the AIFactions.

**generateGeneralOffers:**At the end of some turns, generates new generals to attach to armies.

**generateMercenaryOffers:**At the end of some turns, generates new mercenary armies to hire.

#### 4.4.6.10.MilitaryManager

Extends Manager class



**Methods:**

**manageSieges:**At the end of a turn, search for siege battle possibilities and deals with them.

**calculateLandBattle:**Calculates battle of two armies.

**calculateSeaBattle:**Calculates battle of two navies.

**calculateSiegeBattle:**Calculates battle of an army attacking a city.

**moveArmy:**Handles army movements in Sicily layer.

**moveNavy:**Handles navy movements in Sicily layer.

**createArmy:**Handles the creation of a new army from a garrison.

**createNavy:**Handles the creation of a new navy from a garrison.

**returnCity:**Handles when a force merges to the city again.

**disbandArmy:** Disbands the parameter army.

**hireMercenary:**Creates the mercenary army that player hired.

#### 4.4.6.11.BuildingManager

Extends Manager class.



**Methods**

**newBuilding:**Constructs a new building to given coordinates and with given alignment. Calls related buildings’ update functions.

**updateConstruction:**Calls passTurn methods of buildings if there isn’t anarchy.

**unlockBuilding:**Unlocks appropriate buildings and locks inappropriate ones.

**upgradeBuilding:**Upgrades the building instance that is represented by x y coordinates. Calls related buildings’ update functions.

**deleteBuilding:**Deletes the building instance that is represented by x y coordinates. Calls related buildings’ update functions.

**buildRoad:**Special method for constructing roads, finds the shortest path between two tiles and connects them with a road.

# 5. Glossary & references

<http://www.worldoftropico.com/us/>

<http://www.simcity.com/>

http://www.pogo.com/games/risk#game http://www.moddb.com/mods/war-of-empires1