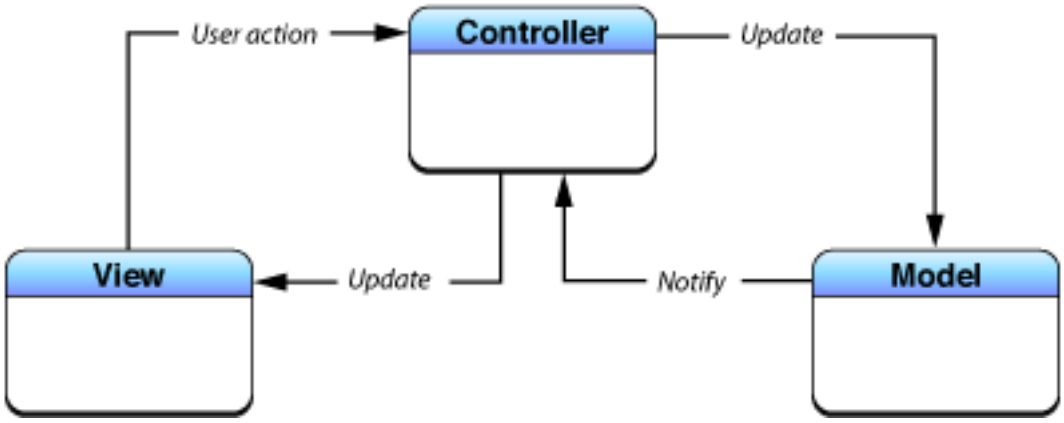
2. SOFTWARE ARCHITECTURE

2.1 Overview

This section describes how our software architecture is. We are making a backgammon game which basically has a UI, takes inputs and puts out outputs, and stores the respective entities inside organised classes. Software world had encountered this problem before and developed architectures styles. We thought that MVC would suit best for our solution. Below, is a very simple diagram of our architectural style:

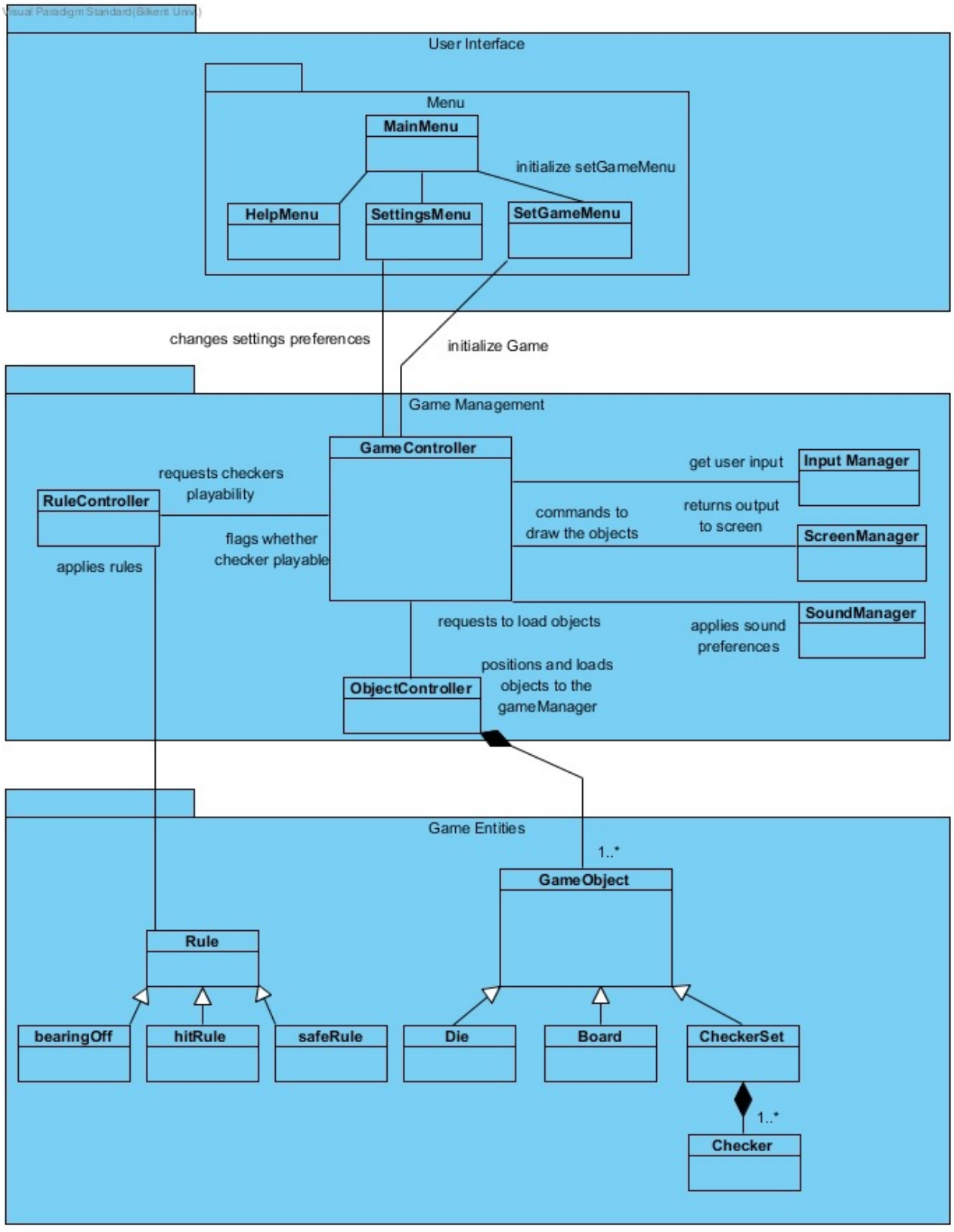


Reasons why we though MVC would suit best are explained in 2.2: Subsystem Decomposition. Note that in the diagram above, associations between subsystems are not hierarchical. In essence, controller works as a intermediary between model and view subsystems.

2.2 Subsystem Decomposition

As mentioned earlier, our system is decomposed with respect to MVC architectural style. Below, is a figure showing details of the decomposition. We sticked to the MVC because our system is divisible as model, view and controller. We could have picked a different architectural style. However, considering good design includes high coherence and low coupling, this can be achieved through putting together our entity classes, manager classes, and UI classes separately. We have tried to put them together and saw that classes in each subsystem runs almost same operations. For instance, RuleController and ObjectController classes, which are in Controller subsystem, modify Rule and GameObject, respectively. This supports the idea of high coherence. Besides that, if we have sticked with MVC style, we can change our model classes (in other words, entity classes that hold information about in-game properties) and that change would obviously effect controller classes. In other words, controller classes' update methods have to change too. We think that this change is acceptable, because it also allows extensive changes to entity classes as well. Obviously, one

can not change the whole game, converting the game from backgammon to chess, but can, for instance, change the implementation of rules to some other format. After that, only respective controller classes about the new format should be changed. No other changes would be necessary unless exceptions. Other architectural styles would cause way more complications than MVC.



2.3 Hardware & Software Mapping

Our system will be coded in Java. And since our game will be very simple, system requirements will also be modest. Hence, any computer that has a decent system properties and an installed java runtime environment can run our program.

Users are going to give inputs using a mouse and keyboard. Trackpad works the same way mouse does, in case the program gets ran on a laptop. Users will see an updated window from the computer's window. So, that means any decent computer will be able to run our program.

2.4 Persistent Data Management

High scores will be saved as XML file. Besides that, sound effect are saved as their individual formats.

2.5 Access Control & Security

Our system is an off-line software. Users will not enter any authentication information, nor they will be entering critical information to the system. Hence, it's security and access control is self-accomplished.

2.6 Boundary Conditions

Initialisation

- JRE is required on the system for our program to initialise and arrive at a playable state.
- If JRE exists, when user opens the application, program starts to execute from where the program is in the disk.

Termination

- User can quit the game at any moment by selecting quit from the menu. Although game process will be lost.

Error

- An unexpected in-game termination of the program will cause a data loss. In other words, game progress will be lost.