Our Design Decisions

* We supported low coupling and high cohesion by dividing classes. In our design, there are many uses Information Expert pattern in the logic layer, i.e; Cell is the information expert of ChewyObject, Board is the information expert of cell, level is the information expert of board. Thus, we have smaller modules in our programs that are easier to change.
* We provided polymorphism using abstract class ChewyObject and interface Matchable. We can add; therefore, a newer types of objects different from lokum with a very small amount of change in our whole system.
* We provide a null object type called Nothing in order to avoid null pointer exceptions.
* We used Model-View-Controller for three of our use cases. GamePlay, LevelSelector and GameLoader are our controllers for use cases Play a Level, Select a Level and Load a saved game, respectively.
* During the first phase, GamePlay’s increasing size with numerous helper methods make this class’ code hard to manage and maintain. Therefore, we decided to create a new class, called BoardUpdater.
* We uses publisher-subscriber for updating the GamePlay’s GUI; by making the GamePlay class a publisher that publishes its state to a list of listeners. Thus, we provided a methodology for updating the UI without violating MVC; and we kept the UI-Logic Layer interaction simple.
* We used Factory of Strategies for the RuleEngine. It was a facade in the first phase. By changing it to a Factory of Strategies and creating different strategy classes, we provided a better modularity and made our system easier to change for varying set of game rules.
* We used singleton for our XML parser, factory and strategy classes; since they are not deeded to have more than one instances at a time.