Projektstruktur  
Projektet består av de följande huvuddelarna:

1. En hemsida som introducerar spelet.
2. En javaapplikation som representerar spelet.

På grund av att hemsidan visar en highscorelista och att spelet behöver många olika frågor skapade vi extra delar som hjälper till att få bra utvecklingsmiljö.  
De två extra delarna är back-enden som är skriven med php och databasen som lagrar vissa data, exempelvis frågorna som ska användas.

För att hämta in frågarna till spelet skapades en ”restful api” som visar frågorna med sina möjliga alternativ i JSON format istället för att ha direktkontakt mellan javaappen och databasen.  
Syftet med det är att man inte behöver utveckla en integrerad back-end i java som hämtar frågorna och sen en annan om vill ha webbbaserad version av spelet.

Databasen är en mysql databas och innehåller de två följande tabellerna:  
- questions: lagrar frågorna och sina alterntaiv som visas till spelare.  
- highscore: lagrar spelares spelomgång.

Följande diagram beskriver projektets struktur.

Hemsidans struktur:

Följande diagram beskriver hemsians struktur.

* index.php: visar hemsidans huvudsida.
* rules.php: visar Regler sidan.
* high\_score.php: visar High-score lista sida.
* info.php: visar Funktionell programmering sida.
* about.info: visar Om Oss sida.
* CSS : mappen innehåller alla cssfiler som används för att styla hemsidan.
* IMG : mappen innehåller alla bilder som används på hemsidan.
* js : mappen innehåller alla javascriptfiler som används på hemsidan.
* Parts : mappen innehåller alla hemsidans små komponenter, exempelvis navigationbar och footern som visas i alla subsidorna.

Spelets struktur: (javaappen)

Följande diagram beskriver spelets struktur.

* UML diagram : mappen innehåller appens UML diagram.
* Javadoc : mappen innehåller appens javadoc.
* Millionaire: mappen innehåller externa bibliotek som användas för att behandla JSON och skaffa mysql-anslutning samt srcmappen.
* src : mappen innehåller mapparna img, sounds, millionaire samt style.css filen.
* img : mappen innehåller alla bilder som används i spelet.
* sounds : mappen innehåller alla audiofiler som används i spelet.
* style.css : har csskoden som stylar vissa komponenter.
* millionaire : mappen innehåller alla javaklasser.
* Model: mappen innehåller alla klasser som tillhör delen ”Model” av MVC.
* View: mappen innehåller alla klasser som tillhör delen ”View” av MVC.
* Controller.java : klassen som representerar delen ”controller” av MVC.
* Main.java : är appens huvudklass.
* Alla andra klasser är beskrivna i javadoc.

Följande umldiagram visar appens klasser och sina innehåll.

Appens designmönster:  
Följande designmönster används i appen:

* Singleton.
* Strategy.
* Model View Controller (MVC).

Singleton:  
Används av vissa klasser som inte ska skapa fler än en instans.  
Exemplevis:

Strategy:  
Används av vissa klasser som är lika men har olika beteende ”Strategy”.  
Exemplevis:

MVC:  
Används som ett huvuddesignmönster som separerar appens delar så det blir lättare och bättre att underhålla och utveckla appen vidare om man vill.  
Vi utvecklade MVC så att man inte behöver många variabler i Kontrollern för att skilja ankommande event. Alltså använde vi inte ”event handler” utan använde lambdaexpression för att anropa vissa metoder direkt istället.

Om vissa saker bara ändras i view när man tex trycker på en knapp ska kontrollern inte blandas in i utan det ska bara tas hand om i view. Men om några ändringar behöver data av modellen ska kontrollern ta hand om det. Dvs skicka signaler till view med datan som behövdes efter att ha hämtat datan från modellen.

Viewen har ingen aning vad modellen har för klasser och vice versa.

Kontrollern kommunicerar bara med modellens huvudklass (Game) av klasserna i modellpaketet.

Kontrollen kommunicerar bara med views huvudklass(Gui) av klasserna i viewpaketet.

Kontrollerns största roll är att vara en koppling mellan viewen och modellen.