Gruppmedlemmar:

Ansvarsområden:

* Henrik: Har ansvarat för mycket dokumentation, hemsidans undersida om funktionell programmering och diverse javakodning som bl.a. livlinor, säkerhetsnivåer och inloggningsskärmen där spelaren skriver sitt namn. Skapade frågorna om Java.
* Jesse: Har ansvarat för hemsidans indexsida, undersidan om reglerna, diverse javakodning som bl.a. livlinorna, pristabellen och fönstret som visas efter spelomgången. Skapade frågorna om Javascript.
* Joakim: Har ansvarat för att göra sidfoten på hemsidan, diverse javakodning som bl.a. poänguträkningen, återställning av spelet och layout. Skapade frågorna om HTML.
* Millad: Har ansvarat för hemsidans undersida om utvecklarna, diverse javakodning som till exempel timerlayout och en del av databaskopplingen.
* Mohammad: Har ansvarat för hemsidans undersida highscore, databashanteringen, strukturen i javadelen och många javadelar som bl.a. frågorna, timern, ljudeffekter, intro och grafik. Har mergat med mastern på GIT och har även skapat UML-diagram och strukturbilderna i dokumentdelen. Skapade frågorna om CSS.

Spelets idé:

Spelaren ska med hjälp av ett javaprogram kunna spela ”Vem vill bli miljonär”. Detta ska ske genom att spelaren i ett userinterface ska få frågor presenterade och klicka på rätt svar. Frågorna ska bli svårare och svårare ju längre upp för ”prisstegen” spelaren klättrar. Frågorna handlar om systemutveckling och är uppdelade i fyra delar(CSS, JAVA, HTML och Javascript).

Spelet har fyra stycken ”livlinor” som är ”50/50” där två av de felaktiga svaren plockas bort, ”byt fråga”, ”fråga publiken” och ”ring en vän”.

Om spelaren klarar alla 15 frågor har man vunnit det maximala beloppet. Spelaren kan alltid välja att stanna även om spelaren läst frågan och då erhålla det beloppet spelaren just då har tjänat in.

Svarar man fel eller att tiden går ut förlorar spelaren alla pengar eller åker ner till närmaste säkerhetsnivå och får enbart dessa pengar. Pengarna förs vidare med namn till databasen som sen hämtas av hemsidan. För fullständiga regler läs avdelningen regler på hemsidan.

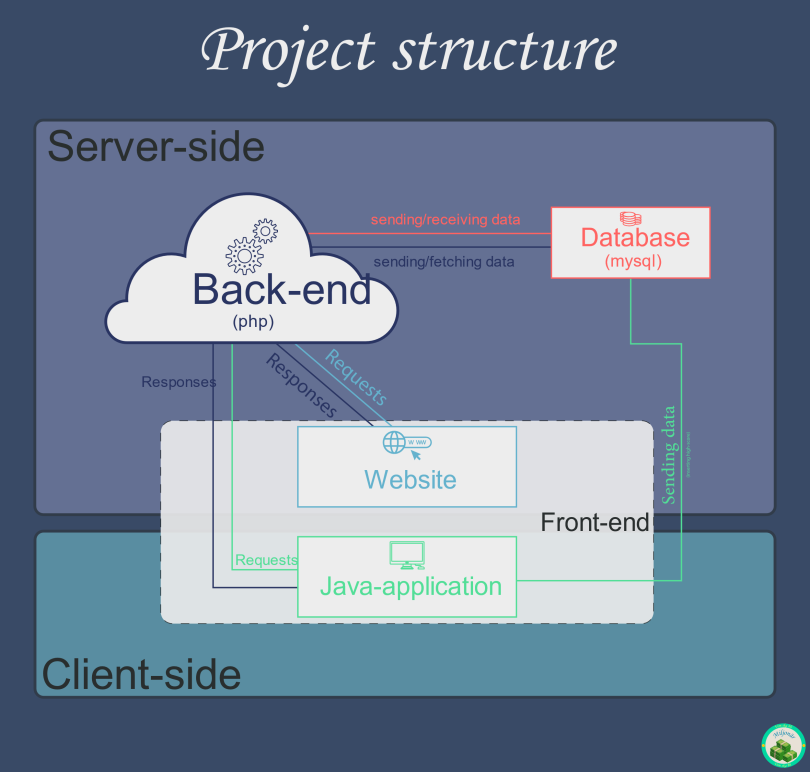
Projektstruktur:  
Projektet består av de följande huvuddelarna:

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-end som är skriven med PHP och databasen som lagrar vissa data, exempelvis frågorna som ska användas.

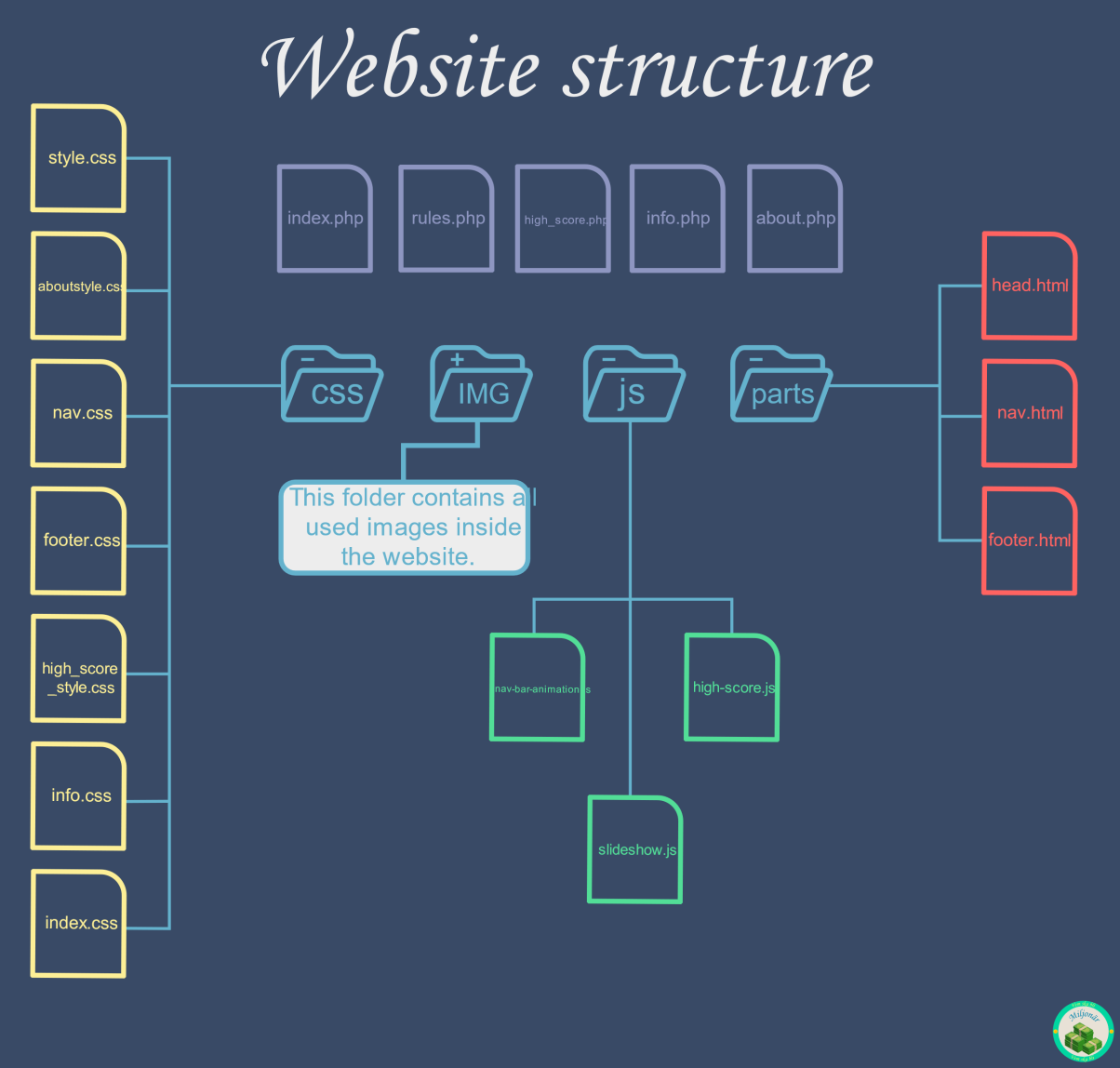
För att hämta in frågorna till spelet skapades en ”restful API” som visar frågorna med sina möjliga alternativ i JSON format istället för att ha direktkontakt mellan javaappen och databasen.  
Syftet med det är att man inte behöver utveckla en integrerad back-end i java som hämtar frågorna och sen en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

Databasen är av typen MYSQL och innehåller de två följande tabellerna:  
- questions: lagrar frågorna och sina alternativ som visas till spelare.  
- highscore: lagrar spelares spelomgång.

Följande diagram beskriver projektets struktur. ”Vissa texter är inte på plats på följande diagram. öppna de själv stående diagram ” SVG filer” för att ha bättre resultat.”

Hemsidans struktur:

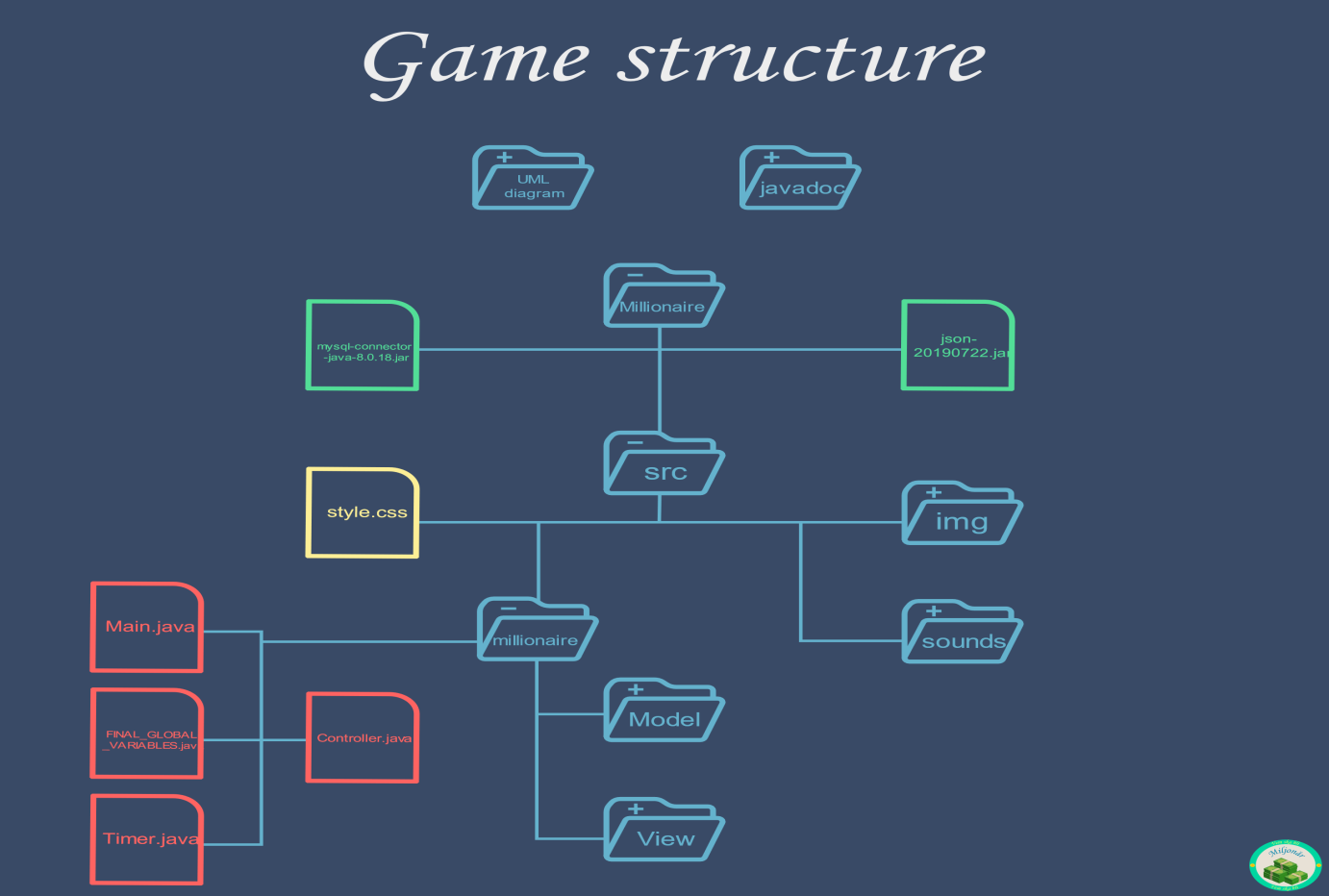
Följande diagram beskriver hemsidans struktur:



* index.php: visar hemsidans huvudsida.
* rules.php: visar sidan om reglerna.
* high\_score.php: visar sidan med highscore.
* info.php: visar sidan om Funktionell programmering.
* about.info: visar sidan om utvecklarna.
* CSS : mappen innehåller alla CSS-filer 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 navigation bar och sidfoten som visas i alla undersidorna.

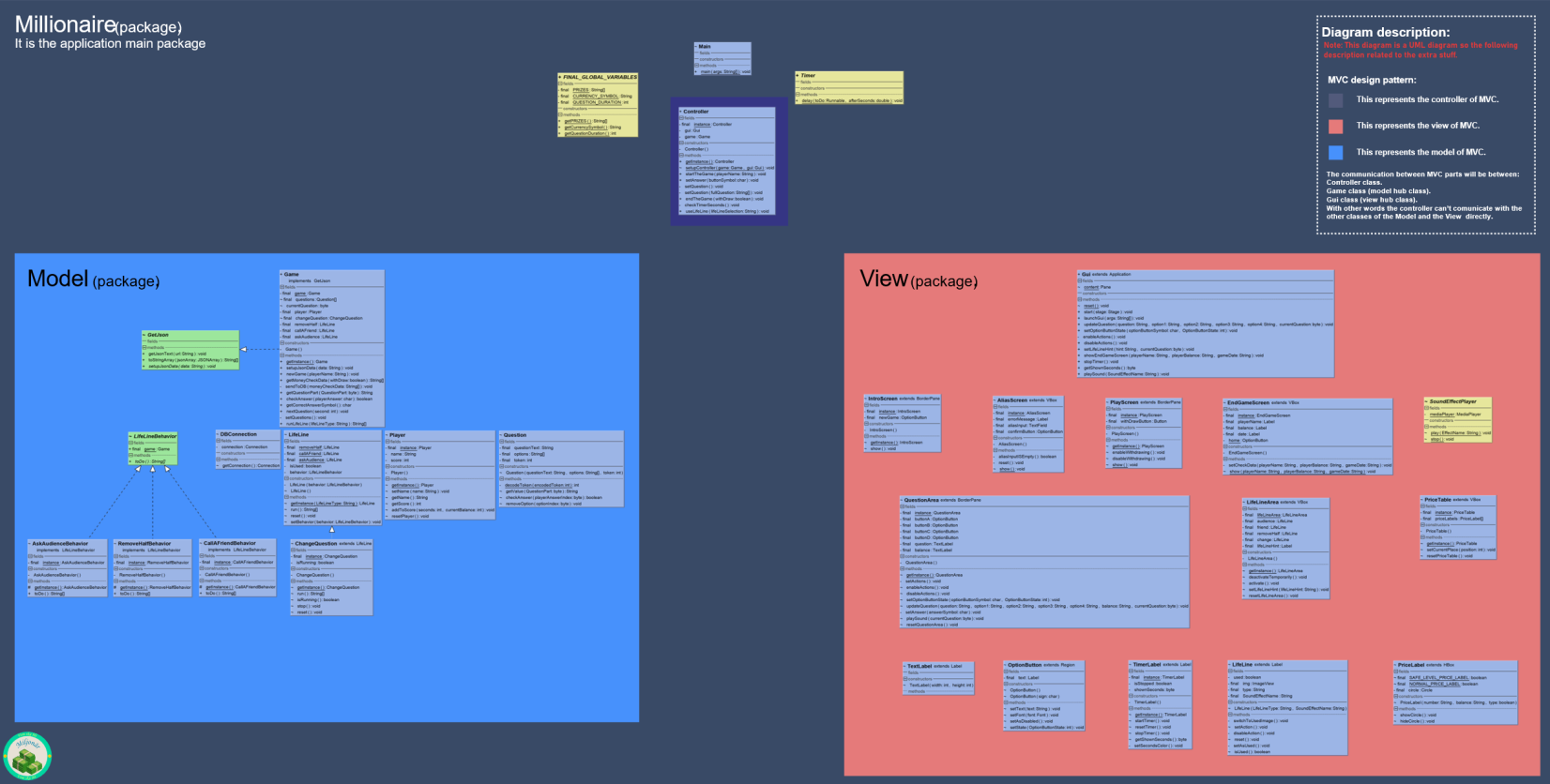
Spelets struktur: (javaappen)

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 databas-anslutning samt mappen src.
* src : mappen innehåller mapparna img, sounds, millionaire samt filen style.css.
* img : mappen innehåller alla bilder som används i spelet.
* sounds : mappen innehåller alla audiofiler som används i spelet.
* style.css : har CSS koden 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 UML-diagram visar appens klasser och deras 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.  
Exempelvis:  
Klassen Game i Model eller klassen PlayScreen i View.

Strategy:  
Används av vissa klasser som är liknande men har olika beteende.  
Exempelvis:  
De tre klasserna som implementerar interfacet LifeLineBehavior.

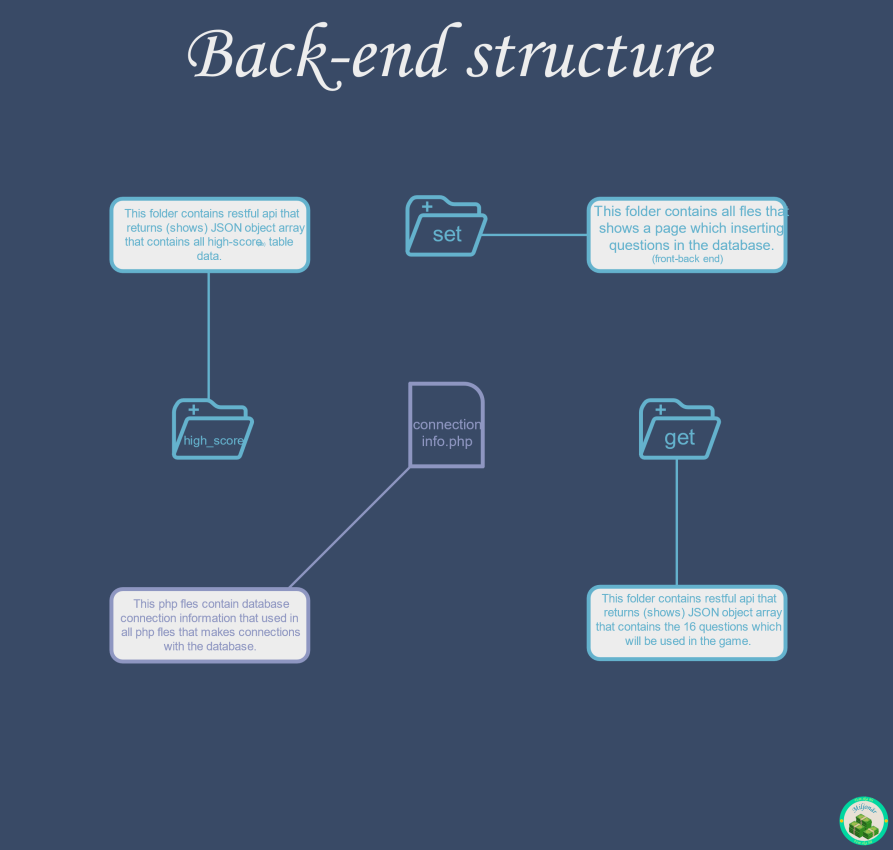
MVC:  
Används som ett huvuddesignmönster som separerar appens delar så det blir lättare och bättre att underhålla samt utveckla appen vidare om man vill.

Vi utvecklade MVC så att man inte behöver så många extra variabler i Kontrollern för att fånga events från View och sedan skicka tillbaka instruktioner till View. Alltså använde vi inte ”eventhandler” utan använde istället lambdaexpression för att anropa vissa metoder direkt från en klass i View till en annan klass i View.

Så om vissa saker bara ändras i View när spelaren till exempel trycker på en knapp ska kontrollern inte blandas in i utan det ska skötas 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.

View har ingen aning vad modellen har för klasser och vice versa.  
Kontrollern kommunicerar bara med huvudklassen(Game) av klasserna i modellpaketet.  
Kontrollen kommunicerar bara med huvudklassen(Gui) av klasserna i View.

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

Back-end struktur:  
Följande diagram beskriver back-end struktur:

Detta diagrammet beskriver back-enden tillräckligt.  
OBS! Alla ovanför filar och mappar måste stå tillsamman med hemsidans filar alltså under samma mappen.

Säkerhet: //TODO

Veckovis utveckling:

Vecka 1:

Alla skapade sina frågor och laddade upp dessa på databasen som Mohammad skapade. Joakim började skissa på hemsidan och Jesse på javadelen. Henrik tog tag i organiseringen.

Vecka 2:

Millad anslöt sent till gruppen och denna vecka bestämde vi att vi skulle använda JavaFX till projektet så alla läste på om detta. Mohammad lade upp projektet på Github och alla anslöt sig till detta. Mohammad utvecklade Jesses javagrund och kopplade denna till databasen.

Vecka 3:

Denna vecka var ren javakodning där alla gjorde metoder till spelet och spelet tog form ordentligt.

Vecka 4:

Mohammad skapade alla bilder som Henrik satte på plats. Jesse fick ordning på pristabellen. Mohammad slutförde så att frågorna och svarsalternativen kom på plats. Samtidigt arbetade allihopa med metoder i Java. Spelet var nu spelbart men det saknades många funktioner.

Vecka 5:

Javakodningen fortlöpte när alla utom Mohammad jobbade med varsin ledtråd. Mohammad väntade in alla andra och arbetade då parallellt med en potentiell webbversion av spelet.

Vecka 6:

Mohammad hjälpte de som kört fast för att gruppen kände att det börjar bli dags att sätta spelet och att vi vill ha klart det nästa eller nästnästa vecka. Mohammad färdigställde även timern och ljudeffekterna. Henrik gjorde klart userinputscreen och Jesse endgamescreen. Millad tog tag i hur spelet skicka spelomgången till databasen.

Vi gjorde även regler för vad som ska gälla för gruppen. Inte för att det hade hänt något utan för att vi glömt det till en börja. Vi planerade även hur hemsidan ska se ut. Denna tog Jesse tag i och började designa denna.

Vecka 7:

Spelet var nu nästintill klart men det återstod många buggfix och metoder för att få allt som vi ville. Millad gjorde en grafisk detalj med timern och arbetade sen med hemsidans aboutsida. Joakim löste metoden som räknar ut poäng och grafiska detaljer med texten. Jesse började med undersidan regler. Henrik tog tag i dokumentdelen och undersidan funktionell programmering. Mohammad fick designa om strukturen då vi upptäckt att denna behöver justeras för att få ett så bra designpattern som möjligt. Han började även på undersidan highscore och hur denna ska läsa in datan från databasen.

Vecka 8:

Alla hade arbetat med sin HTML-del. Millad hade arbetat med aboutsidan, Henrik med funktionell programmering, Jesse med regelsidan, Joakim med sidfoten och Mohammad med highscoresidan. Mohammad hade även strukturerat om klasserna för att ännu bättre passa MVC.

Vecka 9:

Alla hade under jullovet arbetat med sin del i Javadoc och sin del av hemsidan. Mohammad hade gjort diverse diagram och skrivit texten som tillhör dessa. En del buggar har även fixats i spelet.

Vecka 10:

Presentation!

Planerade utvecklingar om vi hade haft mer tid:

* Skapa funktion för att spela många samtidigt.
* Skapa en skärm som visas när programmet hämtar frågorna från databasen.
* Skapa fler frågekategorier spelaren kan välja mellan och/eller en blandning av alla kategorier.
* Skapa en skärm på slutet som i händelse av att spelaren tappat uppkoppling kan få information om detta. I nuläget avslutas spelet korrekt i avsaknad av uppkoppling men ingen poäng skickas ju då till databasen.
* Skapa en webbaserad version av spelet.

Backlogen:

Javadelen:

* Leta upp ljudfiler.
* Börja bygga grunden.
* Spara poäng för att sedan kunna lägga till i databas.
* Kommentera för JavaDoc.
* Bygga GUI med Swing.
* Versionshantera med GIT.

HTMLdelen:

* Funktion för att läsa från databasen och presentera datan.
* Funktion för att läsa reglerna och förstå hur highscorelistan konstrueras.
* Del som beskriver vad funktionell programmering är.
* Versionshantera med GIT.
* Göra hemsidan responsiv.
* Ha bra design på spelet och hemsidan.

Databasen:

* Skapa databasgrunden.
* Skapa hemsida där vi kan addera frågor.
* Skapa phpsidan för att hämta frågorna.
* Skapa highscoretabellen i databasen.

Spelplanering:

* Beskriva kortfattat vad spelet ska gå ut på.
* Komma på 15 frågor per person om varsitt ämne.

Inlämning:

* Inlämning back-end.
* Inlämning front-end.
* Inlämning Worddokument.
* Inlämning Powerpoint.
* Dubbelkolla kraven som Rickard gett oss.

Dokumentdelen:

* Organisera Trello.
* Skapa Messengergrupp.
* Veckovis log: Vem , vad?
* Backlog(Initiala versionen).
* Förklaring av spelet.
* Gruppmedlemmar.
* Ansvarsområde för varje medlem.
* Struktur av spel: klasser, metoder, variabler. Ska visas med Bilder och text.
* Varför valde vi och hur ser vårat designmönster ut? MVC?
* Struktur av hemsida.
* Vilken säkerhet har vi och vilken säkerhet skulle vi vilja ha?
* Vad är XSS(Cross-site Scripting).
* Vilka funktioner skulle vi vilja ha om det fanns mer tid?
* Utvärdering. Bra, dåligt. +-!?. GIT?

Presentation:

* Gå igenom dokumentets punkter.
* Demonstrera projektet.
* Fördelar/nackdelar med CMS.

Utvärdering av projektet:

+ Det vi gjorde bra var att vi klarade av alla problem som uppstod och vi är mycket nöjda med resultatet. Eftersom vi vill att spelet ska ha ett beteende som liknar originalet hade vi varit missnöjda om vi inte hade löst det eftersom det då synts tydligt i resultatet.

+ Vi samarbetade bra och om någon körde fast hjälpte vi varandra. Vi kommunicerade bra på Messenger och använde Trello och GIT. Stämningen i gruppen har varit väldigt god och vi har skrattat mycket.

- Det vi kunde gjort bättre var att redan från början planera exakt hur vårt MVC skulle vara organiserat och hur strukturen skulle vara.

- En annan sak vi kunde ha gjort är att från början skriva ner alla klasser och metoder som skulle behövas men det hade varit svårt och enligt oss var det denna gång bra att addera klasser och metoder allt eftersom.

- Vi kunde skrivit regler innan vi började ifall konflikter hade uppstått. Vi kunde även gjort klart för alla att det är bättre i ett tidigt skede säga att man inte klarar en uppgift.

! GIT fungerade till slut ganska bra. Det var svårt i början men ju mer träning man fick desto bättre gick det. Man lärde sig även med tiden att skriva bra kommentarer när man ”commitade” och att inte ”committa” för ofta. Då detta var första gången vi använde GIT och alla var osäkra så arbetade vi på det sättet att alla pushade upp sin egen bransch och Mohammad mergade alltid dessa med branschen Master. På så sätt minimerade vi konflikter i koden.

! MVC lärde vi oss mer om ju mer kunskap vi fick av Ulf och Rickard.

! JavaFX lärde vi oss då vi upptäckte att Swing inte skulle kunna göra vårt userinterface lika snyggt.

! Javadoc lärde vi oss.

! Då alla i gruppen låg och ligger på olika kunskapsnivåer gjorde vi olika genombrott på olika områden. Till exempel objektorientering och responsiva hemsidor.

? Vi ville göra en jarfil av vårt spel men det lyckades vi inte med. Det finns problem med detta när man använder JavaFx.