# Projektbeskrivning

## TypeRacer
## 2024-03-22

**Projektmedlemmar:**
Enzo Visser <enzcu445@student.liu.se>

**Handledare:**
Thea Antonson <thean981@student.liu.se>

## Innehåll

# Projektplan

## 1. Introduktion till projektet

Jag kommer att utveckla ett ”Typeracer” program där man ska skriva av en given text så snabbt som möjligt, med så lite fel som möjligt. Detta är ett program man kan använda sig till för att förbättra sin skrifthastighet med ett tangentbord men även för att utmana sig själv eller andra för skojs skull.

Den givna texten kommer bestå av en slumpgenerad mening av ett visst antal ord som användaren kommer att kunna modifiera till hens preferens. Dessa modifikationer inkluderar antal ord och specifika boskstäver som orden består av.

Efter att man skrivit av en mening kommer ens resultat att visas vilket kommer i form av tid, cpm (characters per minute) och tills sist ens noggranhet i procent. Denna statistik kommer sedan sparas och sammanställas tillsammans med framtida resultat. Dessa kommer visas i ett fält som även säger om man har gjort bättre eller sämre ifrån sig i förhållande till tidigare resultat.

Det program som jag tagit insperation av är: https://www.keybr.com/.

## 2. Ytterligare bakgrundsinformation

Tiden för varje runda startar när man tryckt i det första tecknet rätt. Om man sedan trycker i fel tangent vid något tecken kommer man inte vidare förrän man har tryckt i rätt. Man kommer inte behöver trycka på backåttangeten när man har fel, man trycker i rätt tecken och fortsätter sedan med resten av meningen.

Cpm (characters per minute/ tecken per minut) räknas ut genom att ta antal tecken i meningen dividerat med den förflutna tiden i minuter.

Sammanställd Cpm räknas sedan ut genom att lägga samman tidigare cpm värden för att sedan dividera dessa med antal värden som lagts samman.

Wpm (words per minute/ ord per minut) använder samma metod som cpm för uträckning och sammanställd uträckning men med skillnaden att man byter ut antal tecken med antal ord eller dividerar antal tecken med en konstant som representerar en ordinarie ordlängd såsom 5.

Noggranheten kommer räknas ut genom att ta alla korrekt angivna tecken dividerat med sammanlagda tecken i meningen multiplicerat med 100 för att få det i procent. Ett felangivet tecken kan alltså endast räknas som ett fel och inte flera för att hålla noggranheten mellan 0-100% och inte få negativa värden. Det kommer alltså inte spela någon roll ifall man anger ett tecken fel 10 gånger i rad, det kommer endast räknas som ett felangivet tecken.

## 3. Milstolpar

| # | Beskrivning |
|---|---|

| | |
|---|---|
| **1** | Skapa ett fönster där programmet ska visas i. Detta innefattar en Viewer och en Launcher som kommer vara min main-class där programmet körs ifrån. |
| **2** | Implimentera en databas av ord i det engelska alfabetet som jag kan använda till att slumpa fram meningar senare. Jag kommer försöka hitta en json lista som innehåller en godtycklig mängd ord. |
| **3** | Skapa en ruta i mitt fönster (antingen en label eller en textpane) där texten senare ska slumpas fram men som jag nu skriver till en placeholder. |
| **4** | Skapa en meningsslumpare som slumpar ett antal ord från min json-lista som sedan visas i den givna rutan. |
| **5** | Se till att texten byter till en ljusare färg då man skriver den givna meningen så man kan se hur mycket av meningen man skrivit. Detta inkluderar även att tecken ska färgas röda om man skriver ditt fel tecken. |
| **6** | Skapa en klocka som håller koll på förlutna tiden under en runda. Denna ska även visas över textrutan. |
| **7** | Se till att hålla koll på antal felangivna tecken och förfluten tid för att sedan räcka ut wpm, cpm och noggranhet som ska visas i en ruta efter varje runda. |
| **8** | Sammanställd statistik ska sparas och visas i toppen av programmet. Alltså ska ens gemonsnittliga wpm, cpm och noggranhet visas i toppanelen med timern. |
| **9** | Skapa en inställningsknapp där man kan ändra på meningen genom att få bestämma antal ord i meningen i ett visst intevall samt om man vill exkludera några tecken från meningen. |
| **10** | Skapa en pauseknapp som ser till att man kan pause en session, då ska klockan pausat samt inputs inte tas emot. |
| **11** | Se till att felhantera samt hantera felen genom att skicka ut en dialogruta. Tex om man ger ogiltiga inställningar eller om resurhantering går dåligt. |
| **12** | Skapa ett visuellt tangentbrod i programmet som visar vilken tangent som trycks ned. |
| **13** | Se till så att man måste logga in som en användare innan man kan använda programmet. Ens sammanställda statisk ska sparas för respektive användare. |
| **14** | Skapa en highscorelista där de högst cpm/wpm värderna sparas och uppdateras. Denna ska även sparas på fil. |
| **15** | Göra det möjlgit att generera kompletta meningar som inte endast består av slumpmässiga ord. Måste därmed ha en lista av kompletta meningar. |
| **16** | Göra det möjligt att lägga till specialtecken såsom ! och ? samt kunna alternera mellan stora och små bokstäver. |

| | |
|---|---|
| **17** | Lägga till AWPM (Adjusted Words per Minute) som tar hänsyn till både wpm samt ens accuracy. Detta är även en bra kandidat till highscorelistan. |
| **18** | Se till att felhantera på de nya implimentationerna. |
| **20** | |
| **21** | |
| **22** | |
| **23** | |
| **…** | |

# 4. Övriga implementationsförberedelser

Orden som meningsgeneratorn kommer ha tillgång till kommer ifrån denna json lista: https://github.com/bevacqua/correcthorse/blob/master/wordlist.json. Som innefattar lite mer än 2200 olika ord av olika storlek i bokstavsordning. Denna lista ska jag hämta från resursmappen i repot.

# 5. Utveckling och samarbete

Jag arbetar själv men jag ska försöka vara i så god tid som möjligt med alla milstolpar samt hinna klart enligt min nuvarande planering.

# 6. Implementationsbeskrivning

## *6.1. Milstolpar*

Mistolpar:

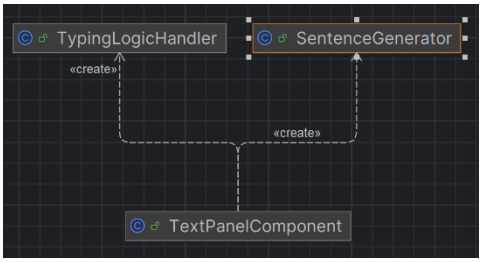
1. Denna milstolpe är genomförd, **SpeedTyperLauncher** är main-klassen som kör programmet genom att skapa en viewer (**SpeedTyperViewer**) samt anropa dess showfunktion som skapar fönstret för programmet.
2. Denna milstolpe är genomförd, Json-listan med ord tas från följande länk: https://github.com/bevacqua/correcthorse/blob/master/wordlist.json, som innehåller lite mer än 2200 godtyckliga ord i varierande storlek, sorterad i bokstavsordning. Jag namngav listan ”wordlist.json” och lade den under resource-mappen.
3. Denna milstolpe är genomförd, **TextPanelComponent** är klassen som visar texten.
4. Denna milstolpe är genomförd, **SentenceGenerator** läser in Json listan och genererar därefter en slumpad mening med orden från listan.
5. Denna milstolpe är genomförd, **TypingLogicHandler** hanterar logiken då man skriver vilket inkluderar färgning av tecken samt förflyttning av markören så man vet var man är i meningen.
6. Denna milstolpe är genomförd, **Timer** är klassen som skapar en klocka och håller koll på tid, **TimerLabel** är klassen som visar tiden i rätt format.
7. Denna milstolpe är delvis genomförd, **TypingLogicHandler** hanterar antal fel under en runda och beräknar noggranhet. Sedan kan **SessionAccuracyStatLabel** visa noggranheten och **SessionCpmStatLabel** kan räkna ut Cpm med hjälp av **Timer** och visa den. Dessa visas sedan i en **SessionStatFrame** när en session är över. Dock så implementerades aldrig en wpm etikett då arbetet bortprioriterades då cpm etiketten redan var implementerad.
8. Denna milstolpe är delvis genomförd av precis samma anledning som den innan.
9. Denna milstolpe är genomförd, **SettingsButton** är knappen för att öppna **SettingsFrame** där man kan ändra på storleken av meningen och om man vill exkludera några tecken.
10. Denna milstolpe är genomförd, **PauseController** hanterar logiken när man pausar medan **PauseButton** ändra pause-tillstånd.
11. Denna milstolpe är genomförd.
12. Denna milstolpe är delvis genomförd, **VisualKeyboardComponent** skapar

det visuella tangentbordet genom att sammanställa tangetkomponenter från **CustomKeyComponent**. Jag hann inte implementera så att alla tangenter på det visuella tangentbordet lyser upp, utan endast de engelska bokstäverna samt siffrorna gör detta.

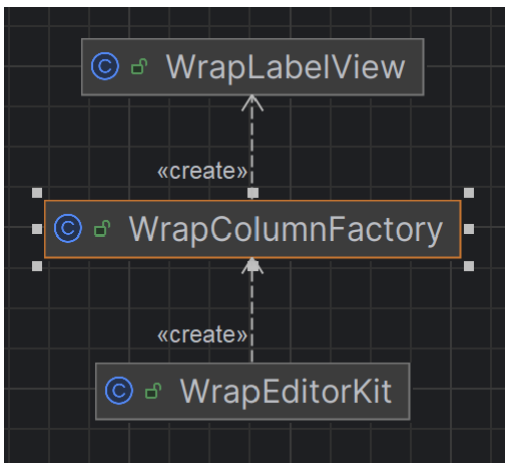
13. Denna milstolpe är inte genomförd på grund av tidsbrist. Det har dock behövts implementera annan funktionalitet såsom **CustomEditorKit, Loggning** och **AbstractTypingException** vilket inte togs till hänsyn i planeringen.
14. -||-
15. -||-
16. -||-
17. -||-
18. -||-

## *6.2. Dokumentation för programstruktur, med UML-diagram*

Main-klassen i mitt program, klassen som startar programmet, är **TypeRacerLauncher. TypeRacerLauncher** skapar en instans av **TypeRacerViewer** som i sin tur skapar instanser av alla visuella komponenter som visas i huvudfönstret när programmet startar. De objekt som **TypeRacerViewer**, skapar är de visuellla komponenterna **TextPanelComponent**, **Timerlabel AvergeAccuracyStatLabel**, **Average CPMStatLabel**, **SessionAccuracyStatLabel**, **SessionCPMStatLabel**, **PauseButton**, **SettingsButton**, **VisualKeyboardComponent** samt de logiska komponenterna **Timer**, **PauseController**, **TypingEventHandler**, som de visuella komponenterna tar del av.




**TextPanelComponent** är den textkomponent som visar upp den mening som ska skrivas av i programmet. Den kan göra detta genom att delvis skapa en instans av en **SentenceGenerator** som generar slumpmässiga meningar baserat på visa typer av inställningar och **TypingLogicHanlder** som hanterar logiken när man skriver. Den tar emot inmatning från användaren, jämför den med den uppvisade meningen och uppdaterar textpanelen för att visa rätt eller fel inmatning.



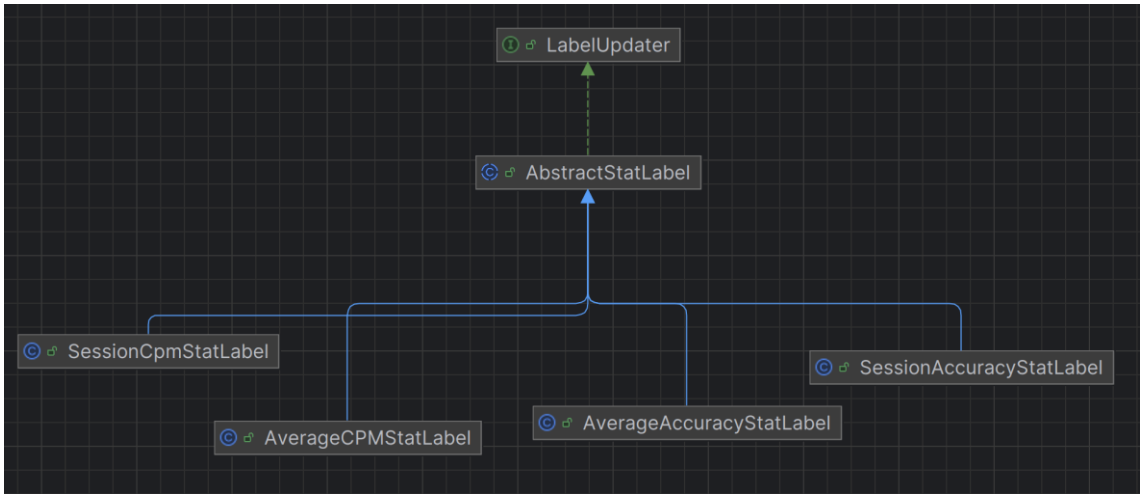

**WrapEditorKit** är en specialiserad “StyleEditorKit” som skapar och använder en **WrapColoumnFactory** för att skapa vyer som stödjer textomslagning, där en vy är en abstrakt representaton av hur en del av ett dukuments innehåll visas på skärmen. Denna är central för att möjliggöra att texten i **TextPanelComponent** kan radbrytas korrekt. **WrapColumnFactory** är som en fabrik som skapar olika typer av vyer baserat på elementtypen. Den skapar och använder sig av **WrapLabelView** för att skapa etikettvyer som stödjer omslagning. **WrapLabelView** är en anpassad implementation av ”LabelView” som stödjer omslagning av text. Den överskrider ”getMinimumSpan” -metoden för att säkerställa

att texten kan omslutas inom sin behållare som i detta fall är **TextPanelComponent**. **TypingLogicHandler** använder sig av **WrapEditorKit** för att säkerställa att texten i **TextPanelComponent** omsluts korrekt.

Programmet innehåller flera klasser som hanterar olika statistiketiketter som alla är baserade på en gemensam abstrakt klass och ett gemensamt gränssnitt.

**LabelUpdater** är ett gränssnitt som definierar en metod för att uppdatera etiketter. De klasser som implementerar detta gränssnitt behöver alltså tillhandahålla sin egen uppdateringslogik. Detta möjliggör polymorfism där olika implementeringar kan behandlar enhetligt.

**AbstractStatLabel** är en abstrakt klass som implementerar **LabelUpdater**-gränssnittet. Den innehåller metoder för att uppdatera etikettens text och



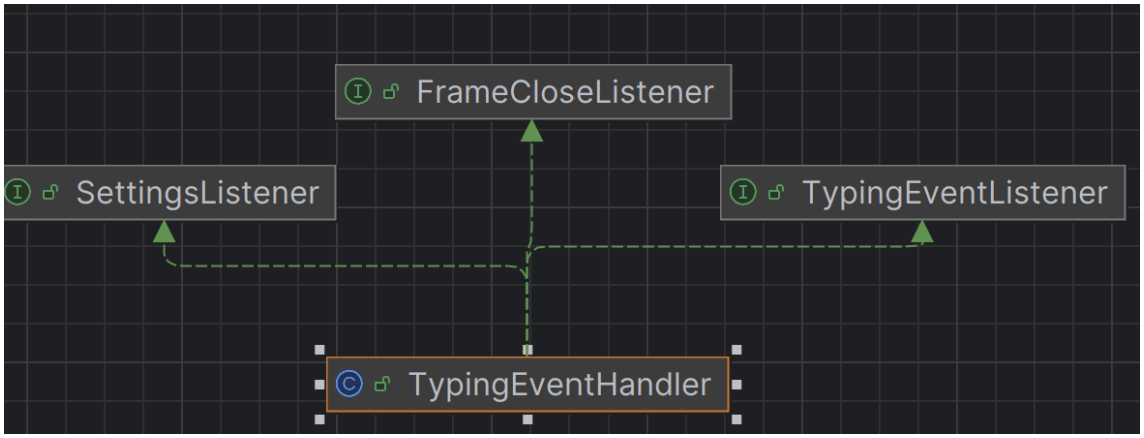

bakgrundsfärgs baserat på statistik som räknas ut i **TypingLogicHandler.** Den tillhandahåller alltså alla gemensamma funktioner för statistikettiketterna. **SessionCpmStatLabel, SessionAccuracyStatLabel, AverageCPMStatLabel** och **AverageAccuracyStatLabel** är alla konkreta implementationer av **AbstractStatLabel.**

**AverageCPMStatLabel** visar användarens sammanställda skrifthastighet i tecken per minut. Detta gör den baserat på data från **TypingLogicHandler** och **Timer**. **AverageAccuracyStatLabel** visar användarens sammanställda noggrannheten i procent. Den hämtar även data från **TypingLogicHanlder** för att göra detta.

De två sessionsettiketerna visar respektive statstik för den aktuella skrivsessionen. Det gör dem via **SessionStatFrame** som skapar en ny ruta med dessa labels efter att en session är klar. För att hålla koll på när händelser inträffar i programmet har jag behövt implementera en del ”listeners” som lyssnar efter specifika händelser och notifierar andra klasser om dem så att de kan reagera följaktiligen.

**FrameCloseListener** är ett gränssnitt som definierar en metod för att svara på händelser när en frame stängs. **TypingEventListener** är ett gränssnitt som definierar metoder för att svara på händelser när skrivning startar och avslutas. **SettingsListener** är ett gränssnitt som definierar en metod för att svara på händelser när inställningar ändras av användaren.

**TypingEventHandler** implementerar dessa gränssnitt för att hantera dessa olika typer av händelser. Den överskrider gränssnittens förrdefineraade metoder för att

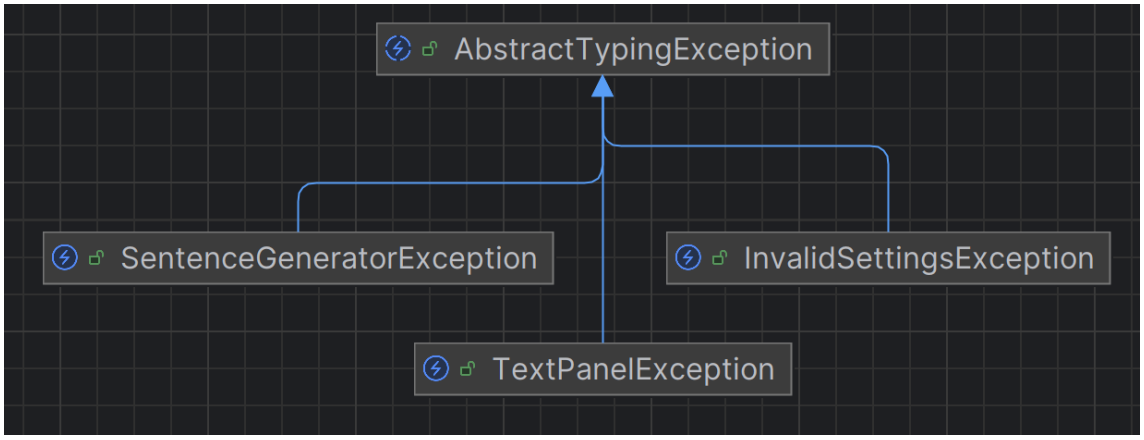



tillhandahålla specifik logik för varje typ av händelse. Ett exempel på detta är att **FrameCloseListener** kommer vara implementerad inom **SessionStatFrame** som kallar på dess definerade metod vilket får **TypingEventHandlers** överskrida metod att kallas. Detta gör det möjligt för en ny mening att genereras så fort **SessionStatFrame** stängs av användaren.

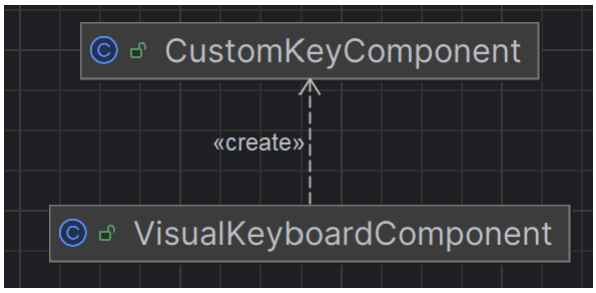
**PauseButton** är den komponent som hanterar paus- och återupptagningsfunktionaliteten i applikationen. Den interagerar med **PauseController** för att växla pausläget och uppdaterar sin etikett (text) baserat på tillståndet. **PauseController** hanterar logiken för att pausa och återuppta applikationens funktionalitet. Den interagerar med **Timer** och **TextPanelComponent** för att kontrollera tillståndet för skrivsessionen. Den uppdaterar timern och skrivhanteraren baserat på om applikationen är pausad eller inte, och loggar dessa åtgärder.

För att hantera fel i programmet används **AbstractTypingException** som är en abstrakt klass som fungerar som en bas för anpassade undantag. Den tillhandahåller konstruktorer för att initera undantaget med ett meddelande och en orsak.

**InvalidSettingsException** används för att signalera ogiltiga inställningar som angivits av användaren i **SettingsViewer**. När användaren anger ogiltiga värden, kastas detta undantag och hanteras för att informera användaren om felet. **TextPanelException** är ett anpassat undantag som representerar fel specifika för textpanelen i applikationen. Och **SentenceGeneratorException** är ett anpassat undantag som representerar fel specifika för meningsgeneratorn i applikationen. Dessa Exceptions utökar alltså **AbstractTypingException**.



**SettingsButton** är den visuella komponent som visar en knapp för att öppna inställningsfönstret. När knappen klickas, skapas och visas en instans av **SettingsViewer**. **SettingsViewer** hanterar det faktiska inställningsfönstret där användaren kan justera inställningar. Den validerar användarens inmatning och använder **InvalidSettingsException** för att hantera ogiltiga värden. När giltiga inställningar sparas, notifierar den **SettingsListener** om ändringarna.




Klasserna **CustomKeyComponent** och **VisualKeyboardComponent** samarbetar för att skapa en visuell representation av ett tangentbord. VisualKeyboardComponent fungerar som en överordnad behållare som hanterar layouten av enskilda CustomKeyComponent-objekt. Varje CustomKeyComponent hanterar sin egen visualisering och interaktion, vilket gör det enkelt att lägga till eller ändra enskilda tangenter utan att påverka den övergripande strukturen.

Inkapsling är ett begrepp inom objekt-orienterad programmering som innebär att kapsla in objektets data och metoder så att de inte är direkt åtkomliga från andra delar av programmet. Istället exponeras nödvändiga funktioner via publika metoder. Detta används i min kod för progammet i samtliga klasser med logisk implementaiton för att skydda objektens interna tillstånd och säkerställa att de endast manipuleras på definierade vis.

# 7. Användarmanual

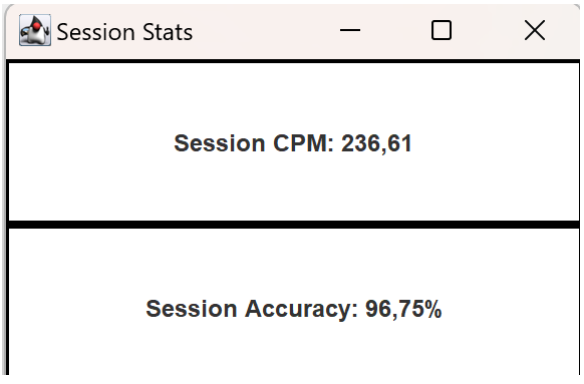
När du startar programmet så kommer följande fönster att visas. En slumpgenererad mening kommer visas i textpanelen i mitten av fönstret. Målet är att skriva av den slumpgenererade meningen så snabbt som möjligt med så få fel som möjligt.

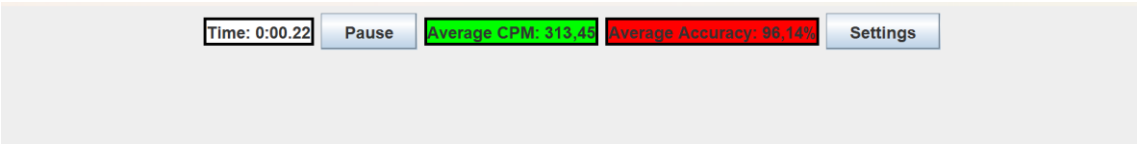
I den övre panelen syns en timer, den håller koll på den förflutna tiden. När du skriver ditt första tecken av meningen så kommer timern att starta. När du väl har skrivit klart meningen så kommer timern att stanna. Under tiden du skriver så kommer tecken att få en ljusare färg för att indikera att de har blivit rätt angivna.

Om du skriver fel tecken under en omgång så kommer det att bli rött. För att sedan skriva vidare på meningen måste du skriva in det korrekta tecknet, då kommer den få den ljusgråa färgen som tidigare rätt angivna tecken.

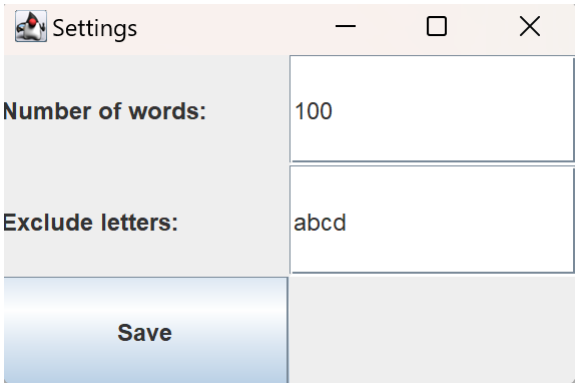
Om du vill pausa timern under en omgång så trycker du på pause-knappen i den övre panelen. Då kan du inte fortsätta din skrivning tills du väljer att klicka på knappen igen vilket kommer visa texten ”Resume” istället för ”Pause”.


Efter att du har skrivit klart meningen så kommer följande fönster att visas. Under Session CPM visas tecken per minut för den omgången. Under Session Accuracy visas din noggranhet i procent. När du stänger detta fönster så kommer en ny mening att genereas i textpanelen.






Efter en omgång kommer även den genomsnittliga statistiken att uppdateras. Om den har förbättras från föregående runda så kommer dess bakgrund bli grön och om den försämras blir den röd.




Om du vill göra några ändringar på den genererade meningen så kan du klicka på settings-knappen. Då kommer ett settings-fönster att visas där du kan ändra på inställningarna. Du kan ändra på antal ord som meningen ska bestå av samt om du vill exkludera några bokstäver från orden som bygger upp meningen. Om du ger en felaktig input kommer ett felmeddelande att visas. Detta felmeddelande kommer även anvisa om vad som var fel med inputen.