Jason Sathya Citro - Project Portfolio

PROJECT: Dukemon

Overview

Dukemon is a desktop flashcard application used to reinforce memory techniques. The user interacts with it using a CLI, and it has a GUI created with JavaFX. It is written in Java, and has about 25 kLoC.

Summary of contributions

- **Major enhancement**:
  - **added Game Result page after game ends**
    - What it does: It shows the user a page to indicate the game is over, which contains the user’s performance on the current game (score, time taken), the user’s progress (progress chart, achievement badges, high score, fastest clear), and a list of words the user has missed to help improve the user’s subsequent performance.
    - Justification: It helps game-ify the process of learning, gives the user a chance to look at how to improve their performance, and also gives a sense of accomplishment for being able to see the progress made after a game session finished.
    - Highlights: The usage of time as a metric of performance (time taken, fastest clear) required understanding on the code for the timer, which led to in-depth discussion with a group mate (Koh Yida).
  - **added Statistics for each word bank**
    - What it does: It shows the user a page on the summary of the statistics of a word bank (can be seen in open mode). It contains the user’s progress (progress chart, achievement badges, high score, fastest clear) and a list of 5 words which the user gets wrong most often.
    - Justification: It gives the user a way to see their progress on a specific word bank, and shows user which words the user is weak at.
  - **added Global Statistics of overall usage**
    - What it does: It shows the user a page on the main page of their overall usage of the app. It contains the user’s total games played, weekly games count, and most played word bank.
    - Justification: By showing the global statistics right on the home page, it makes the app feel more personalized. Additionally, showing the total games played and weekly games count may increase user’s app usage, as each game is recorded and it feels like the user is making their own footprint on the app.

https://github.com/AY1920S1-CS2103T-T11-2/main/commits?author=jascxx **[Link to codebase]**

- **Other contributions**:
    - Refactored the models of AB3 (Person, AddressBook, and their closely-connected classes) to correctly fit the app (Card, WordBank, etc) in the very early stage of the project, which leads to increased productivity from the other members, as it was one of the biggest bottleneck of the project's progress.
    - Sourced for and integrated the avatars for Dukemon that provide a more fun and flavorful learning experience [credit to Geovanny Gavilanes].
    - Created the logo for the app.

Contributions to the User Guide

Given below are sections I contributed to the User Guide. They showcase my ability to write documentation targeting end-users.

WordBank Statistics Commands 📊

(Available in Open mode)

Resetting the *WordBank* statistics: `reset`

Resets the statistics of the open *WordBank* to an empty statistics.

Format: `reset`

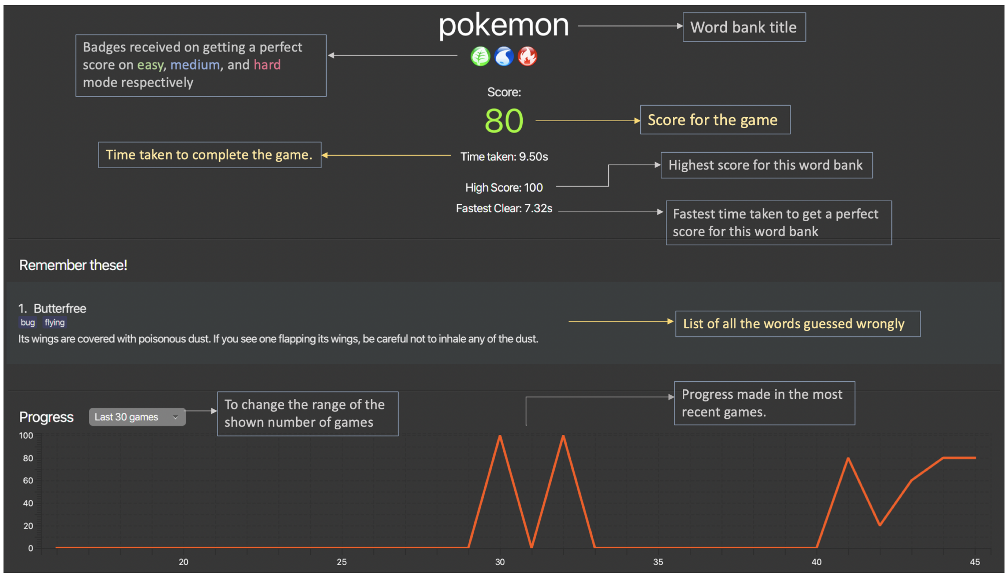
Statistics 📊

This section covers the statistics shown to the user.

Game Result

The game result is shown to the user every time they finish a game. It contains information of the finished game and some information of the *WordBank*.

We use a simple formula to calculate the score: *floor of (Number of correct answers) / (Total questions) * 100*.


Word bank title
Badges received on getting a perfect score on easy, medium, and hard mode respectively
Score for the game
Time taken to complete the game.
Highest score for this word bank
Fastest time taken to get a perfect score for this word bank
List of all the words guessed wrongly
To change the range of the shown number of games
Progress made in the most recent games.
Last 30 games
100
80
60
40
20
0
20
25
30
35
40
45

pokemon

Score:

80

Time taken: 9.50s

High Score: 100

Fastest Clear: 7.32s

Remember these!

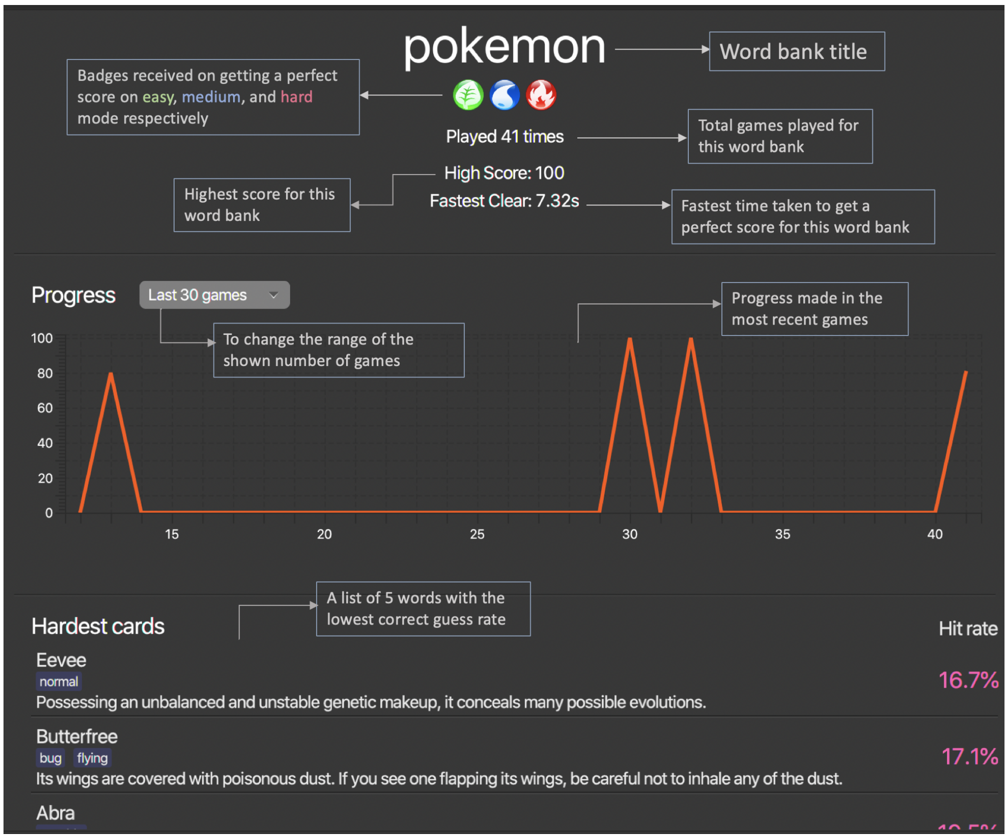
1. Butterfree
bug flying
Its wings are covered with poisonous dust. If you see one flapping its wings, be careful not to inhale any of the dust.

Progress

Figure 1. Game Result UI.

WordBank Statistics

The *WordBank* statistics is shown on open mode and contains all information of the *WordBank*.


Word bank title
Badges received on getting a perfect score on easy, medium, and hard mode respectively
Total games played for this word bank
Highest score for this word bank
Fastest time taken to get a perfect score for this word bank
To change the range of the shown number of games
Progress made in the most recent games
A list of 5 words with the lowest correct guess rate
Last 30 games
100
80
60
40
20
0
15
20
25
30
35
40

pokemon

Played 41 times

High Score: 100

Fastest Clear: 7.32s

Progress

Hardest cards	Hit rate
Eevee normal Possessing an unbalanced and unstable genetic makeup, it conceals many possible evolutions.	16.7%
Butterfree bug flying Its wings are covered with poisonous dust. If you see one flapping its wings, be careful not to inhale any of the dust.	17.1%
Abra	

Figure 2. Wordbank Statistics UI.

NOTE The high score and fastest clear of a word bank will not be reset on adding/deleting cards.

Global Statistics

The global statistics is shown on the main title page and contains all information regarding the user's overall usage of the app.

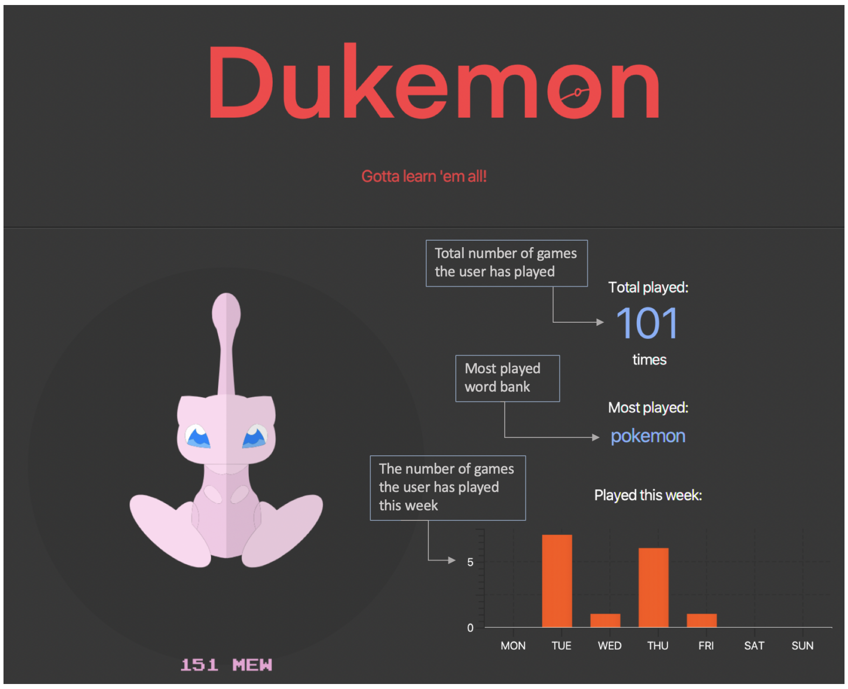


Figure 3. Global Statistics UI.

Contributions to the Developer Guide

Given below are sections I contributed to the Developer Guide. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.

Statistics component

The Statistics component includes 2 main subcomponents:

- A `GlobalStatistics`, containing the user's total number of games played and the number of games played in the current week.
- A `WordBankStatisticsList`, which is a collection of `WordBankStatistics`, one for each `WordBank`.

The class diagram of the Statistics component is shown below:

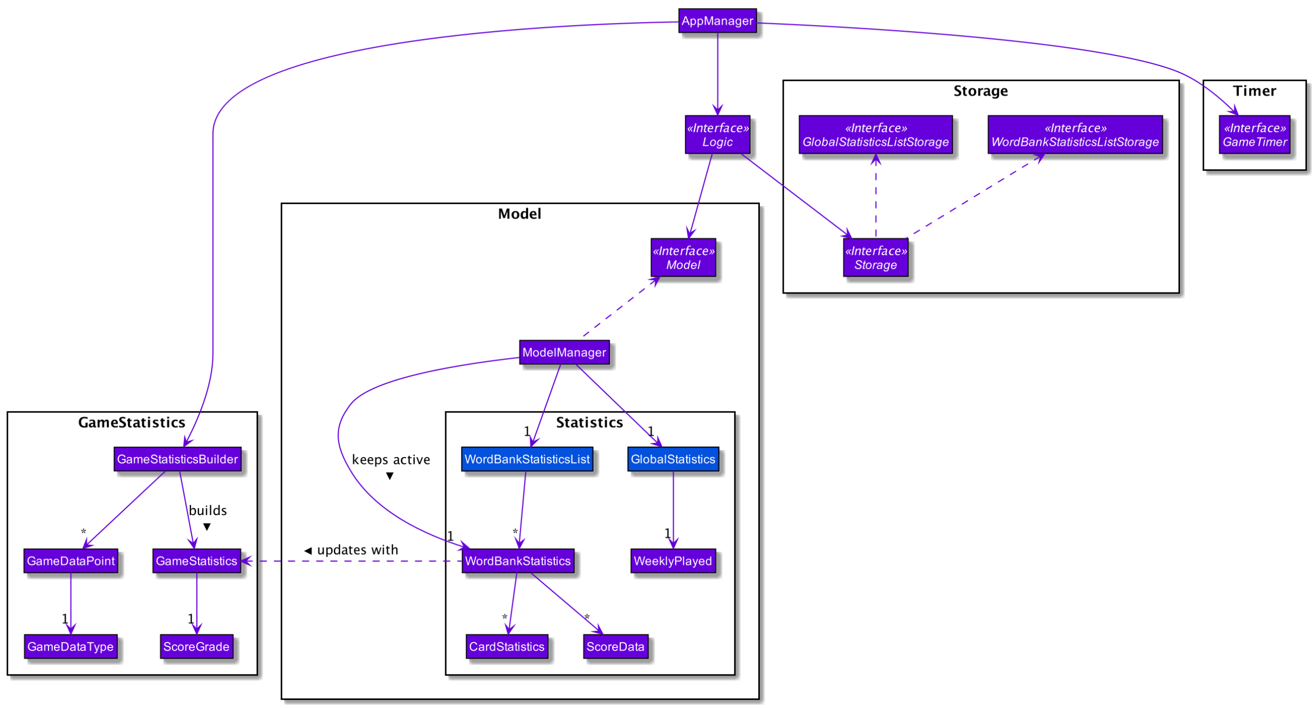


Figure 4. Statistics class diagram.

Statistics features

Implementation

The work of the Statistics component can be neatly captured and explained using a common series of user actions when operating the app.

User action	Statistics work	UI Statistics updates
User opens the app.	User's `GlobalStatistics` and `WordBankStatisticsList` are loaded into `Model` by the `MainApp`.	User is shown their `GlobalStatistics` and their most played word bank from the `WordBankStatisticsList` in the main title page.
User selects a word bank.	The selected `WordBankStatistics` from the `WordBankStatisticsList` is loaded into `Model`.	
User opens the selected word bank.		In open mode, User is shown the `WordBankStatistics` of the opened word bank.
User plays the game.	A `GameStatisticsBuilder` is used to record user actions during the game.	

User action	Statistics work	UI Statistics updates
User finishes the game.	• A `GameStatistics` is created from the `GameStatisticsBuilder`. • The `WordBankStatistics` and `GlobalStatistics` are updated accordingly and saved to disk.	`GameStatistics` and the corresponding `WordBankStatistics` are displayed to user in the game result page.

We will discuss each step with its implementation details primarily on the statistics work.

1. User opens the app

When the user opens the app, their `GlobalStatistics` and `WordBankStatisticsList` are loaded into `Model` by `MainApp`.

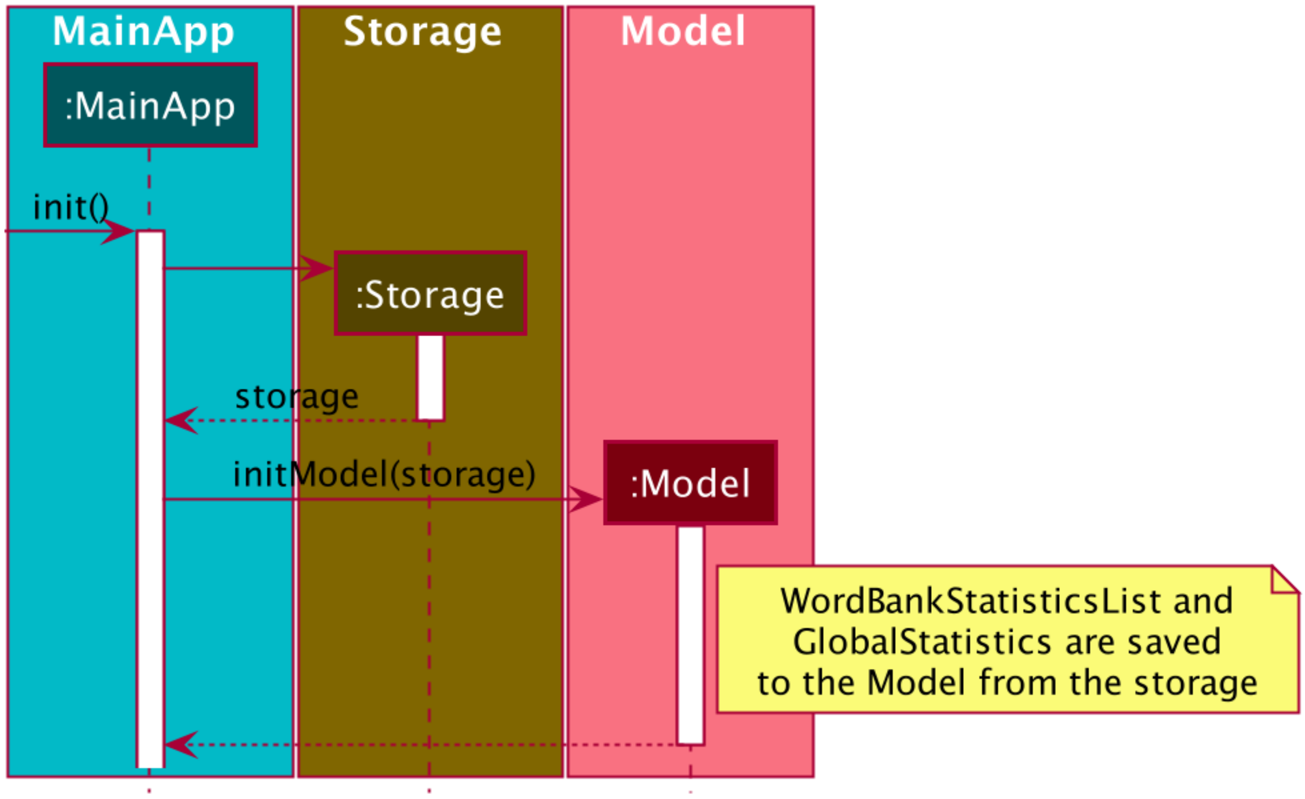

MainApp
Storage
Model
:MainApp
init()
:Storage
storage
initModel(storage)
:Model
WordBankStatisticsList and GlobalStatistics are saved to the Model from the storage

Figure 5. Sequence diagram for loading statistics

2. User selects a word bank

When the user selects a word bank, the selected `WordBankStatistics` from the `WordBankStatisticsList` is loaded into Model.

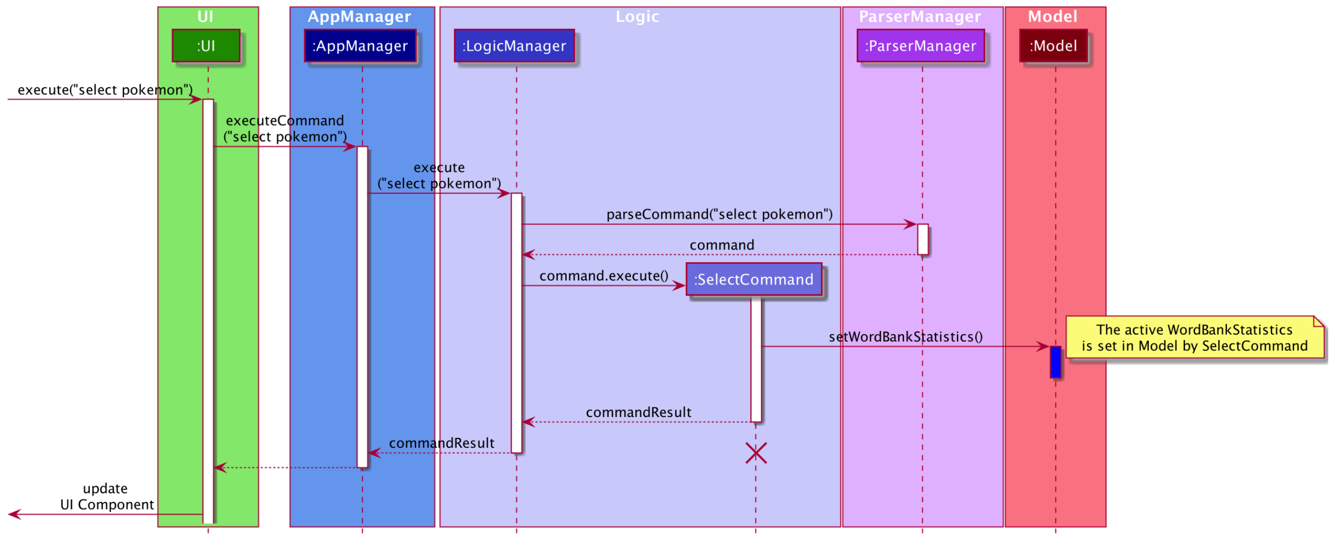


Figure 6. Sequence diagram for selecting a word bank statistics.

It is necessary to set the active `WordBankStatistics` in the `Model` such that when the user opens the `WordBank`, the `WordBankStatistics` can be found in `Model` and shown in the UI.

3. User opens the selected word bank

In open mode, the user is shown the `WordBankStatistics` of the opened word bank, which is set in `Model` at step 2.

4. User plays the game

A `GameStatisticsBuilder` is used to record user actions during the game.

When the user starts the game by calling a `StartCommand`, the `GameStatisticsBuilder` is initialized. Additionally, the `GameStatisticsBuilder` is updated with every `GuessCommand` or `SkipCommand` made during the game. It receives the timestamp from the `GameTimer` which also resides in `AppManager`.

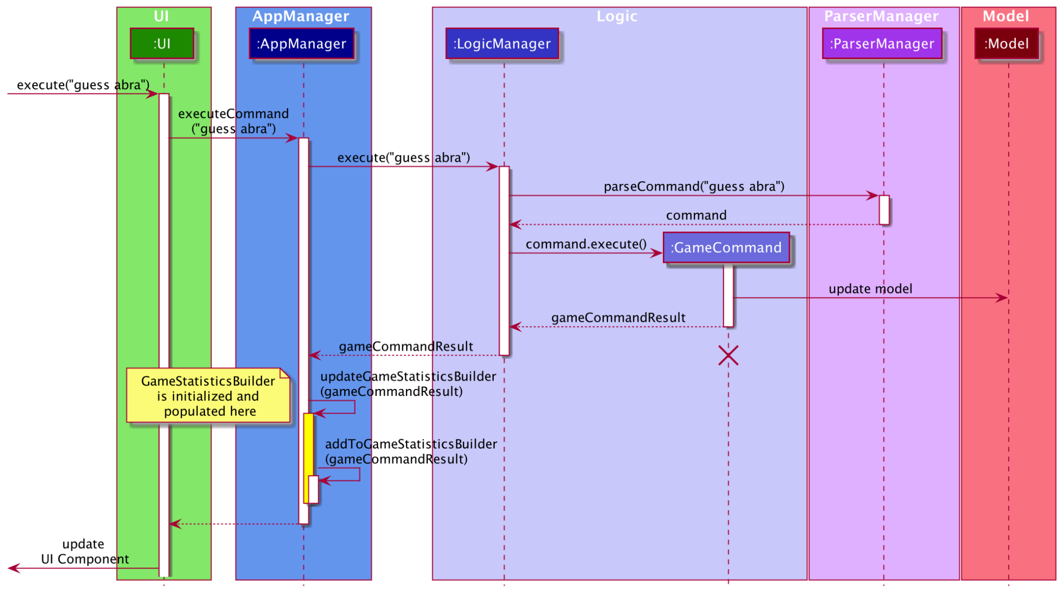


Figure 7. Sequence diagram when user makes a guess.

5. User finishes the game

When the user finishes the game, a `GameStatistics` is created from the `GameStatisticsBuilder`. The `GameStatistics` is shown to the user in the game result page.

The `GameStatistics` is used to update its corresponding `WordBankStatistics`, which is then saved to disk. Additionally, the `GlobalStatistics` is also updated and saved to disk.

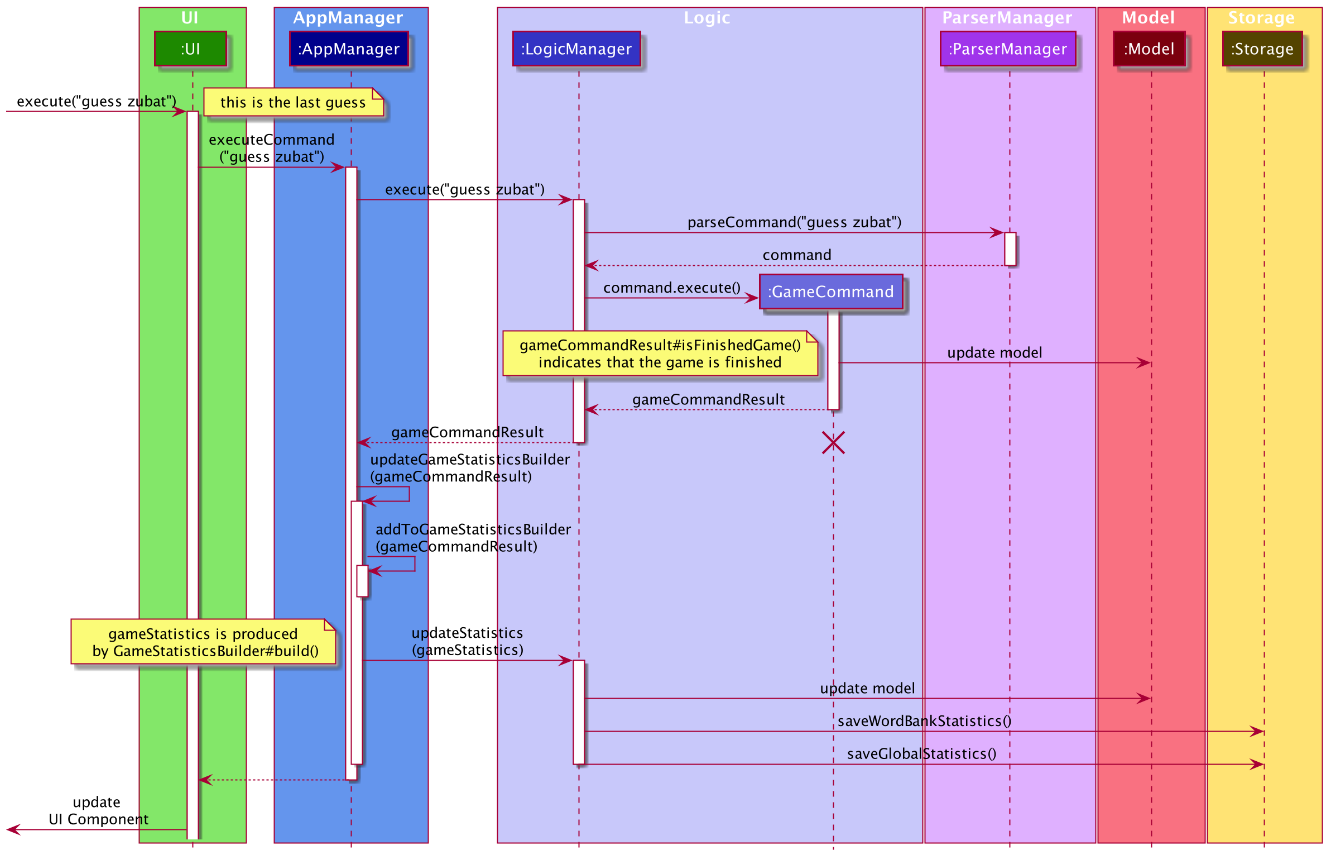


Figure 8. Sequence diagram when the user makes the final guess.

The work done in step 4 and 5 is executed in `AppManager` and the checks to decide what to do are done in the same method `updateGameStatisticsBuilder(CommandResult)`.

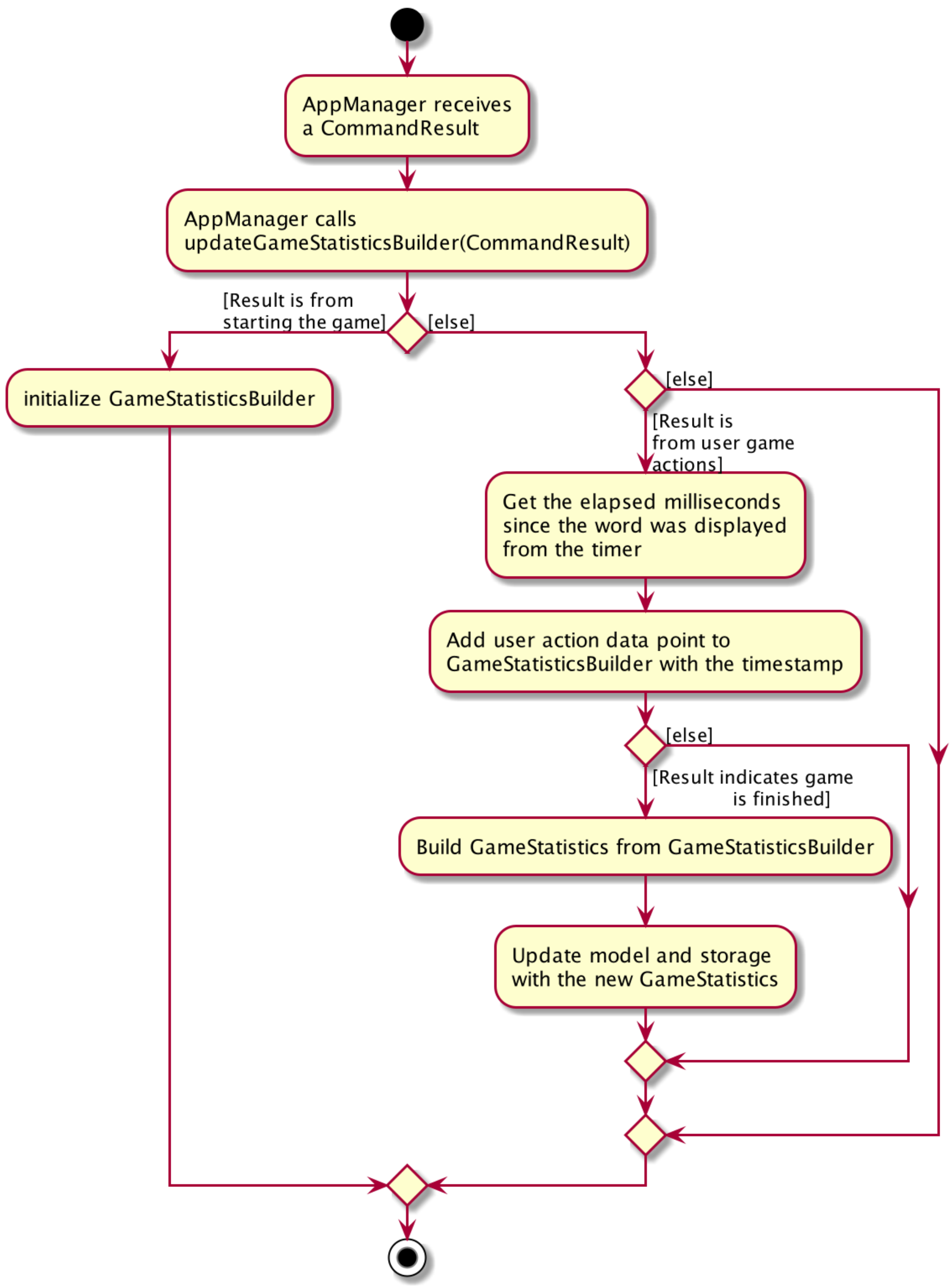

AppManager receives a CommandResult
AppManager calls updateGameStatisticsBuilder(CommandResult)
[Result is from starting the game]
[else]
initialize GameStatisticsBuilder
[else]
[Result is from user game actions]
Get the elapsed milliseconds since the word was displayed from the timer
Add user action data point to GameStatisticsBuilder with the timestamp
[else]
[Result indicates game is finished]
Build GameStatistics from GameStatisticsBuilder
Update model and storage with the new GameStatistics

Figure 9. Activity diagram when `AppManager` receives a `CommandResult` (Details unrelated to statistics are omitted).

Design Considerations

There were some design considerations on implementing the statistics.

	Alternative 1	**Alternative 2**

Aspect 1: How to store `WordBankStatistics` in the storage?	**Store in a separate file from the `WordBank` json file, but with the same name in a different directory.** Example: `WordBank` data is stored at *data/wordbanks/pokemon.json* while the `WordBankStatistics` data is stored at *data/wbstats/pokemon.json* *Pros:* More abstraction to separate the data. *Cons:* The data is linked by name, so if the user changes the file name, the link is broken.	**Store `WordBankStatistics` data in the same file as `WordBank`** *Pros:* Less number of files. *Cons:* Data is combined into one which lowers abstraction.
Why we decided to choose Alternative 1: We decided that abstraction between the data is important as each team member should work in parallel, such that it is easier for one person to modify the storage system for the word bank and another person to modify the storage system for the word bank statistics freely.		