

**Software Development**

**Problem Based Learning**

**“Morra” Game**

**Group J**

Course Title: Higher Diploma in Science in Computing (Software Development)

Date of Submission: 08/08/2020

Lecturer: Adriana E. Chis

|  |  |
| --- | --- |
| Students: | Michele Di Benedetto x19189141 |

# Assignment Description

This application allows 2 users to play the “Java Words Game” which is a turn-based game, where the players take turns to provide one word at a time according to the rules of the game.

At the start of a game, each of the two users is given N lives, and each of them starts the game with zero points. The game should display the number of lives the players received. In addition, the game should inform the players how they can acquire points

# Requirements Summary

A game is formed from multiple rounds.   
Round rules:

1. At the start of each round, the game (i.e. computer) randomly selects one letter from the English alphabet and displays the letter to the players (note that there are 26 letters in the English alphabet).
2. First, the first player enters a word that begins with the letter which has been randomly selected at the beginning of that round. The word has to be formed from at least 3 letters. Input validation is required according to the rules.
3. Second, the second player enters a word that begins with the last two letters of the last word provided by the other player. The word has to be formed from at least 3 letters. Input validation is required according to the rules.
4. Next turn every player needs to respect the same rule explained in Rule No3.
5. The round continues with the players taking turns and entering a word according to the rules specified above. Each time a player provides a valid word, the player receives points according to the rule for awarding points.
6. A round ends when a player is not able to provide a word that begins with the last two letters of the word previously entered by the other player. A player shows that he/she cannot provide such a word when he/she enters a “-“. Note that the player may not be able to provide a word either because he/she does not know a word that begins with the last two letters of the last word provided by the other player or because such a word may not exist (i.e. there is no word that begins with those 2 letters). At that time, the player loses one life, and the other player will start the next round.

The game is played as described above until one of the players loses all the lives allocated by the game. The winner of the game is the user who still has lives left at the end of the game.

At the end of the game, the game displays who the winner is, and the number of points received by each player.

# Rules assigned to the Student

## Number of lives

* Player number of lives: at the start of a game, each of the two users is given 3 lives.
* The player can try to guess a word as many times as the player wants, as per basic requirement the only way to skip a turn is to type “-”

## Rule applied based on my student number 19189141

### Rule to award points

Rule 4 - R4 (penultimate student ID number 4).

The player receives the same amount of points as the number of duplicated vowels  
in the word.

EX:  
 “moon” – 2 points

“cheerleader” – 4 points

“answer” – 0 points

Rule to play Multiple games approach (1 or 3 or 5 or 7 or 9)

Rule - A2 (antepenultimate student ID number 1)  
Ask the players after each game whether they would like to play another game.   
If they answer yes, the game should start again, otherwise the application ends.

# 

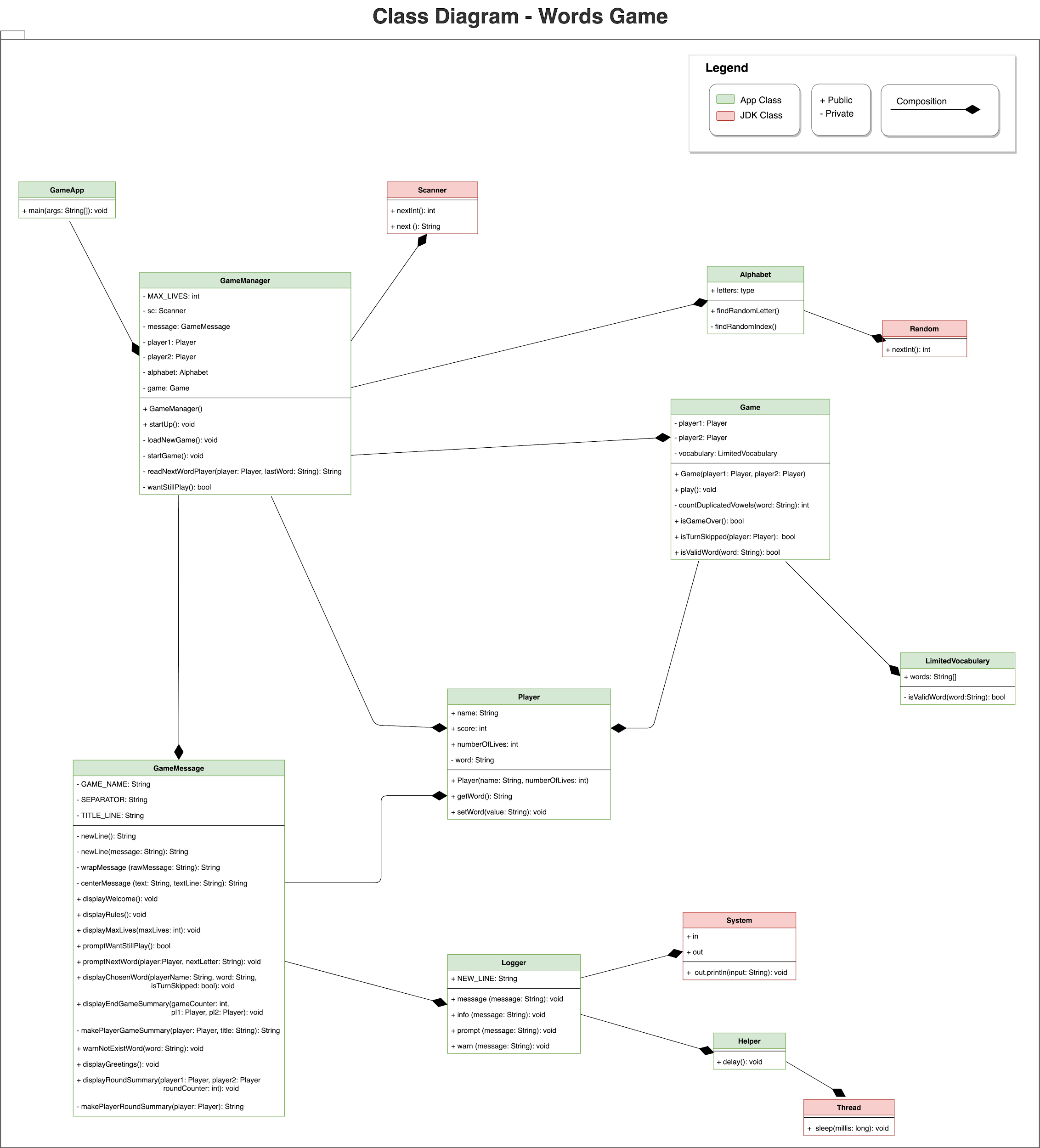
# Design and Implementation Decisions

The program is broken down in several different classes, the idea is to apply single responsibility approach, every class represents one specific use case such us:

* Class **GameApp** contains only the entry point method **main()** and it is only used to start the program and to bootstrap the game lifecycle.
* Class **Player** deals only with task related to the player, is made to be very simple, the only business logic applied is the one to decrease the player lives
* Class **Game** deals only with player and game rules, it does not need to know how the output will be displayed, it gets an input, it processes it and returns something. This class is an ideal place to actually play the game, check if it is game over and validate the game rules.
* Class **GameMessage** deals only with the output, it only prints messages on the console such as a warning, prompting the user for something and showing general information.
* Class **Alphabet** as the name suggests it needs to deal only with English Alphabet and nothing else.
* Class **LimitedVocabulary** deals only with a limited vocabulary provided to guess a word and it validates against its internal array of words.
* Class **GameManager** it act as orchestrator between the other classes, it collect the player input by using class **Scanner**, it process it by using class **Game,**  it show the output by using class **GameMessage,**  it pick random letters by using class **Alphabet,** it sets player attributes by using class **Player.** It does not deal directly with class **LimitedVocabulary,** since the word validation is delegated to the class **Game.**
* Class **Logger** is a helper class, was created to help the class **GameMessage** to creating prefixed message such us "INFO: " (indicates information), ">>>>: " (prompt the user to input something) and "WARN: " (indicates warning about something). Those prefixed strings will help the player to understand what kind of message the console displays
* Class **Helper** was introduced to improve the readability of the console, it relies on the Java class **Thread,** it was introduced to slow down the message printing on the console, as user I feel that printing a bunch of message at once is hard to ready, a delay of 500ms help the user to navigate through those message on the console.

# Class Diagram

This design class diagram shows the list of classes which composites the application. Specifically, there are 9 application classes and 4 JDK library classes.

[](https://www.draw.io/?page-id=C5RBs43oDa-KdzZeNtuy&scale=auto#G10O3s5ZfbU-3CAmdpKBKzQzy7ugEd2oAw)

# Input, Main processing and Output (IPO)

## JDK classes:

1. **Scanner -** used to take the input from the user.
2. **Random -** used to display a random number.
3. **System** -used to display the message on the console.
4. **Thread** - used to sleep the thread enough time to allow the player to read an output console 1 by 1. Not vital for the program.

## Application classes:

### Class GameApp

Contains the *main* method that is used to run the application.

It creates an instance of ***GameManager*** class to bootstrap the game.

### Class GameManager

This class is the core component of the game, it acts as orchestrator between all classes of the application. It is in charge to collect the input of the user, to validate it, to process the input, and show the output.

#### List of class attributes:

* *final int* ***MAX\_LIVES*** *= 3;* It sets the max lives for every player
* ***Scanner*** *sc = null;* It used to handle the user input
* ***GameMessage*** *message;* It used to handle the output of the program
* ***Player*** *player1, player2;* It used to represent the players and their attributes
* ***Alphabet*** *alphabet;* it used to provide a random letter from the English alphabet
* ***Game*** *game;* it used to process the input and compute the output which later will be shown.

#### List of class methods:

* **startUp**() - this method is used to manage the entire lifecycle of the application.
* **loadNewGame**() - it creates the player and game instances. This method is called before every game so that we are sure that any previous data has been reset.   
  N.B. If the game would have handled also a game history this approach would have been different. A reset method into the class Player would have been likely more suitable.
* **startGame**() - it handles the life cycle of a game and its rounds; it takes care of the turn over between players;   
  It reads the input of the player;  
  It displays a round summary after every round;  
  It keeps running until one of the user runs out all lives
* **readNextWordPlayer**(Player player, String lastWord)   
  - it handles the player input and it applies the validation rules
* **wantStillPlay() -** it handles the user wish to play another game by typing “yes” (or y) to play again or anything else to end the game session

### Class Player

This class represents the player of the game.

#### List of class attributes:

* *String name -* Player name
* *int score -* points collected
* *int numberOfLives* - number of lives left
* *String word* - it stores the last word used to play

#### List of class methods:

* getWord() - it is getter for the private member **word**
* setWord() - it is setter for the private member **word** , it plays an important role since decrease the number of lives by -1 in case the last word input is “-”

### Class Game

This class is a core component, it is in charge to process the input of the player and to compute the output.

#### List of class attributes:

* **LimitedVocabulary** vocabulary - It handles the entire English vocabulary user to play the game.
* **Player** player1, player2; - It used to read and process the input of the players. Those 2 objects are injected from outside of this class. Since they are reference variables, once they are instantiated, they can work between **Game** and **GameManager** without the need to be passed as params in every method of **Game** class which need such computation related to them.

#### List of class methods:

* **play**() - It sets the score for both players
* **countDuplicatedVowels**() - compute the point gained by every player based on the *double vowel* rule applied
* **isGameOver() -** it checks if any user runs out the lives, so it returns a boolean to indicate its game is over.
* **isTurnSkipped()** - it checks if the player word indicates to skip the current turn
* **isValidWord()** - it validates the player word against the game rules

### Class GameMessage

This class handles the entire output process inside the game, such welcome message,   
round summary, warning, prompt for input and display who win the game

List of class attributes:

All class attribute names are self-explanatory

* String **GAME\_NAME**
* String **SEPARATOR**
* String **TITLE\_LINE**

#### List of class methods:

All class method names are self-explanatory

* **newLine() / newLine(String message)**
* **wrapMessage(String rawMessage)**
* **centerMessage(String text, String textLine)**
* **displayWelcome()**
* **displayRules()**
* **displayMaxLives(int maxLives)**
* **promptWantStillPlay()**
* **promptNextWord(Player player, String nextLetter)**
* **displayChosenWord( String playerName, String word)**
* **infoWellDone()**
* **displayEndGameSummary( int gameCounter, Player player1)**
* **makePlayerGameSummary(Player player, String title)**
* **warnNotExistWord(String word)**
* **displayGreetings()**
* **displayRoundSummary( Player player1, Player player2**
* **makePlayerRoundSummary(Player player)**

### Class LimitedVocabulary

This class handles a limited vocabulary with 1024 English words encoded as a 1D array of words (source: <https://gist.github.com/deekayen/4148741>). I added a few more words to make the game more engaging.

#### List of class attributes:

All class attribute names are self-explanatory

* String[] words

#### List of class methods:

* **isValidWord**(String word) - it checks if a word is part of the array **words** so is valid word for the game

### Class Alphabet

This class represents the English Alphabet.

#### List of class attributes:

* **char[] letters -** it contains all the letters included in English Alphabet

#### List of class methods:

* **findRandomLetter**() - it provides a random letter from the array **letters**
* **findRandomIndex**() - it provides a random index between 0 and 25 used to pick alter from the array **letters**

### Class Logger

This class is a helper used to show messages more user friendly by prefixing a string with a special word which indicate what the message is for such us:

* "INFO: ": indicates information
* ">>>>: ": prompt the user to input something
* "WARN: ": indicates warning about something

It works strictly close to the class GameMessage

#### List of class attributes:

All class attributes names are self-explanatory

* final String **NEW\_LINE** = "\r\n";

#### List of class methods:

All class method names are self-explanatory

* **message**()
* **info()**
* **prompt()**
* **warn()**

### Class Helper

This class contains helper methods.

#### List of class methods:

* **delay() -** It is used by the **Logger** class to delay the output process in a way that the player has enough time to read a console output before to see the next one into the console. It is not a vital component of the program, but it helps to make the output more readable.

# Build process

The Classes should be compiled in this order:

1. cd ./src
2. javac ./GameApp.java
3. java GameApp

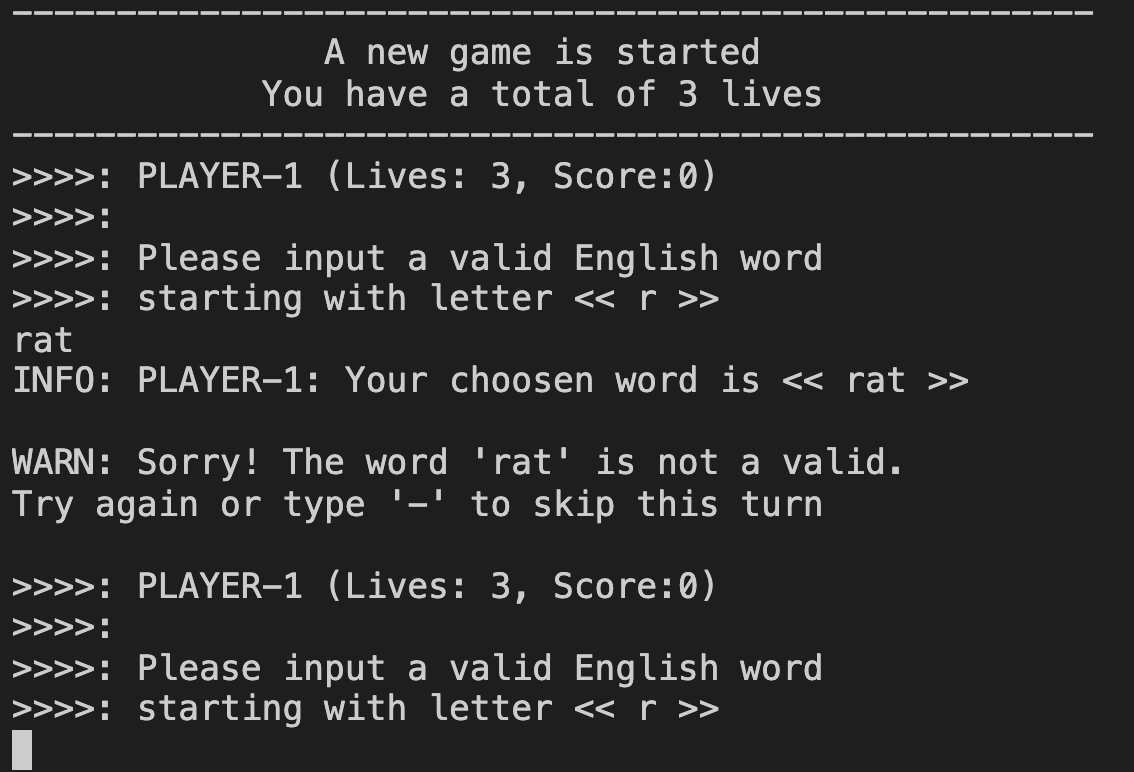
# 

# Manual test

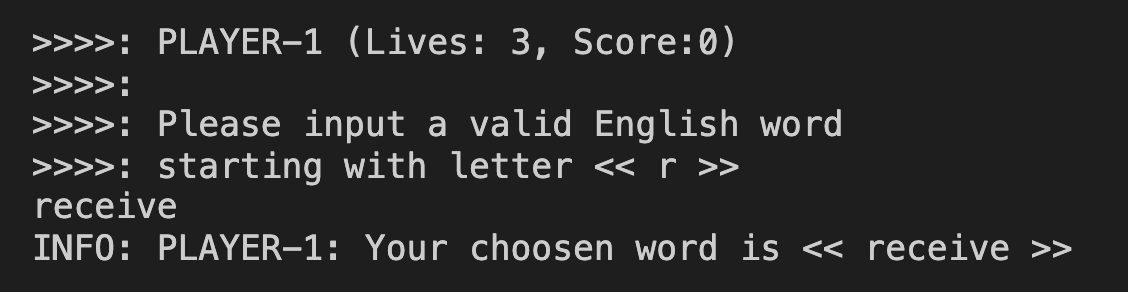
1. The app shows a welcome message and a summary of the rules. A second message asks the Player-1 to input the first word starting with the letter “r”



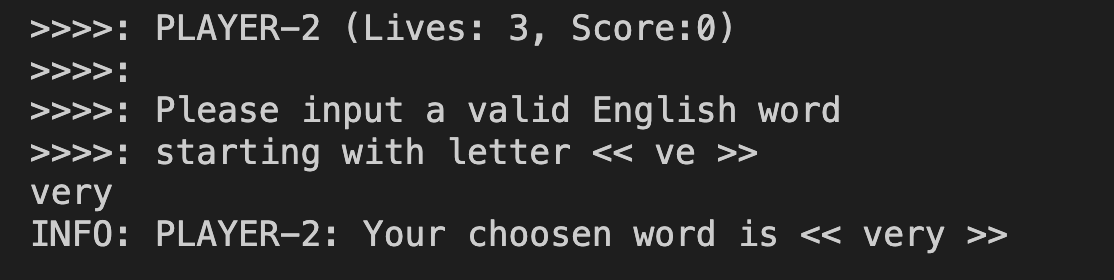
1. Player-1 types “rat” which is not included into the limited vocabulary, then the program displays a warning, Player-1 can either try again or type “-” to skip the turn.



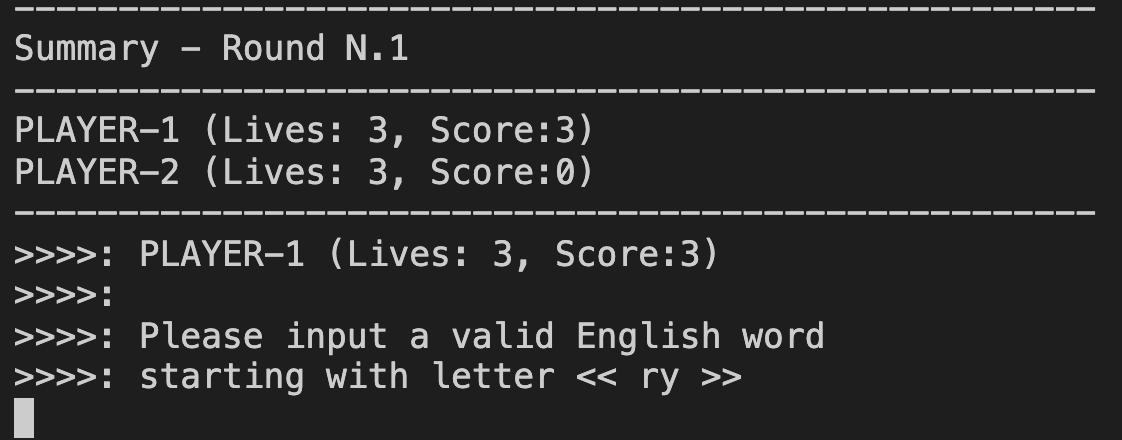
1. Player-1 types “receive“ which is valid word



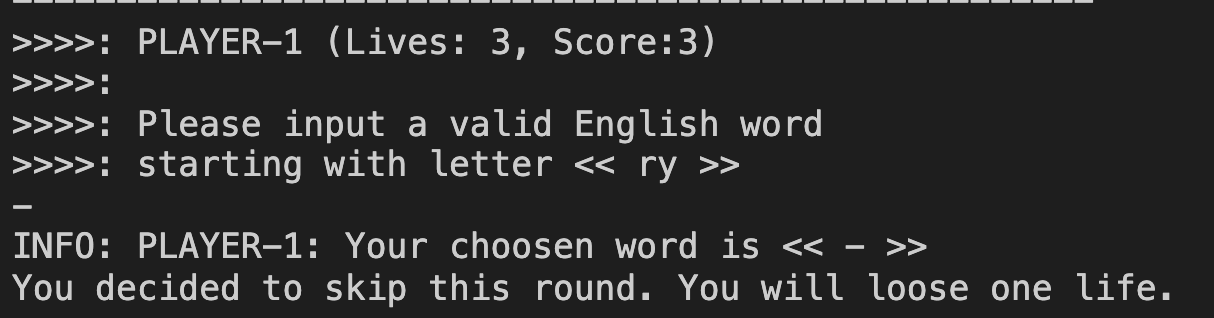
1. Now is the turn of Player-2. The program asks to type a word which starts with the last 2 letters of the previous word, “ve”. Player-2 types “very” which is valid word.



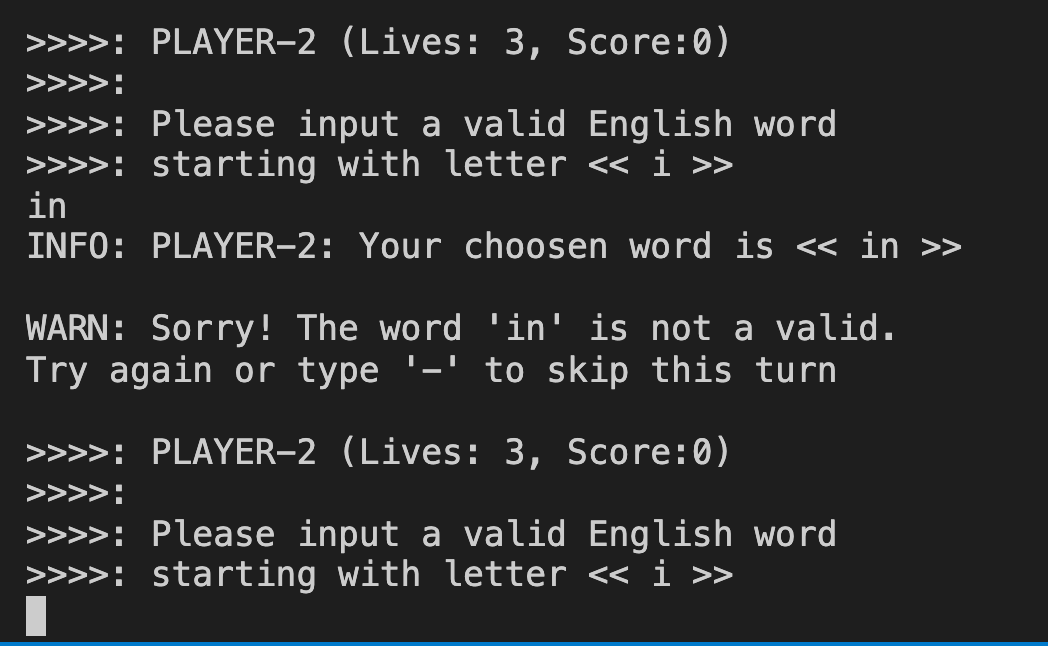
1. The first round is over, the program shows a status summary, the second starts and is again the turn of Player-1 which has to type a word which starts with “ry”.



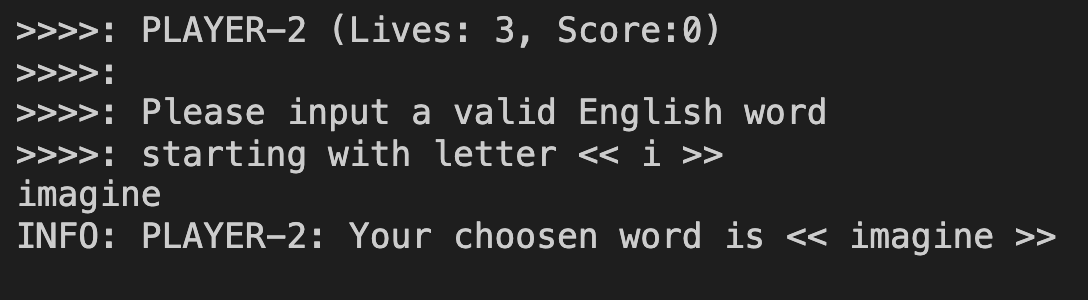
1. Player-1 is not able to guess a valid word so decides to skip this turn, then types “-” to move on. Player-1 loses a life.



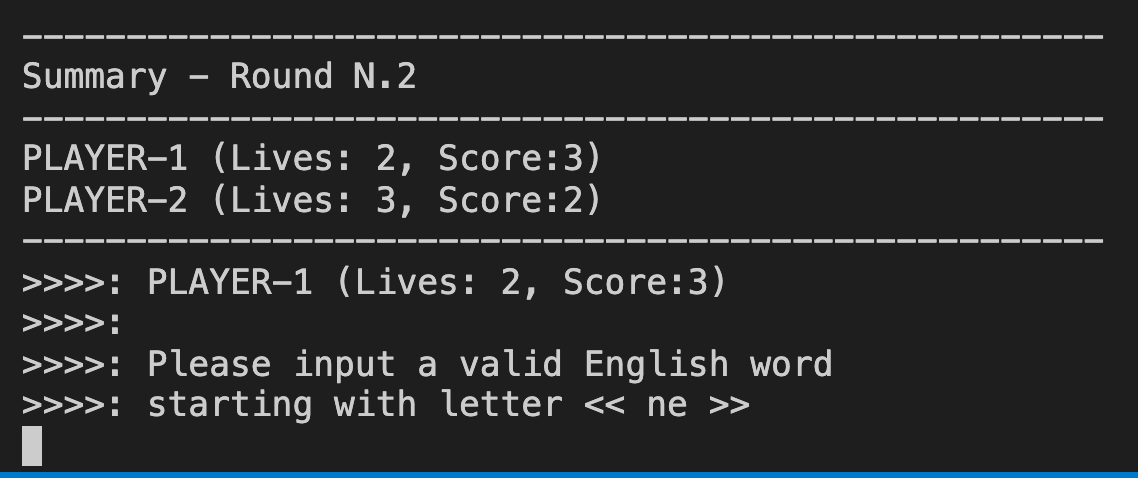
1. Is now Player-2 turn, who is asked to type a word starting with the letter “i”. Player-2 type “in”, the word is not valid, the program displays a warning which states to choose another word or skip the turn.



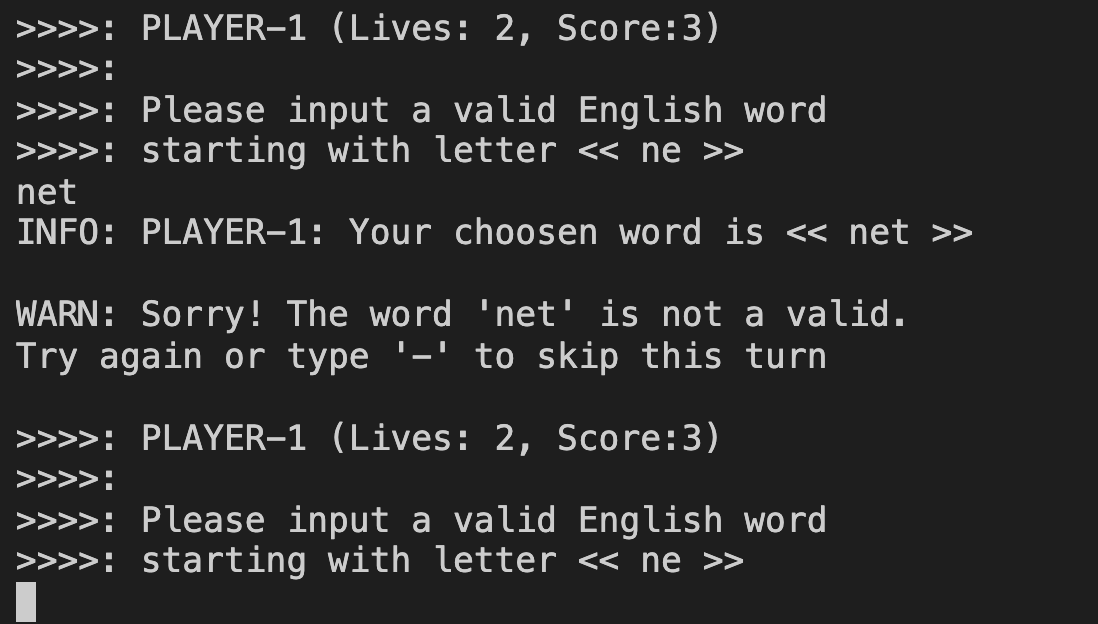
1. Player-2 types “imagine” which is a valid word that passes the turn.



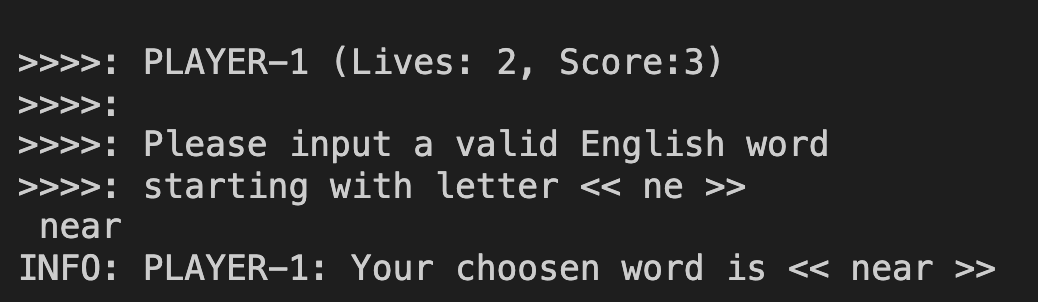
1. The program shows the summary for Round 2. Is now the PLayer-1 turn.   
   Player-1 has to choose a word starting with the letters “ne”



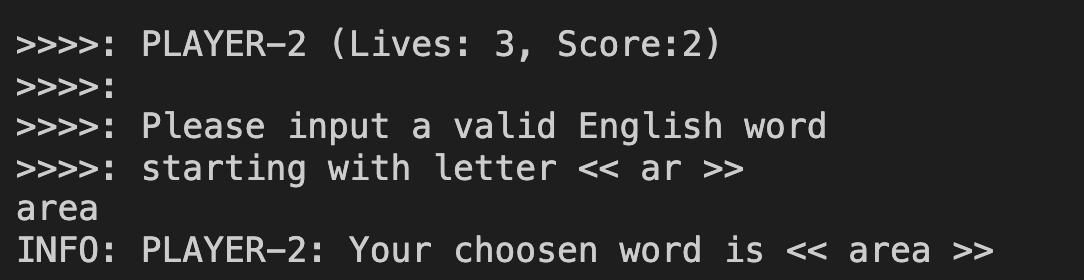
1. Player-1 types “net” which is a word not included into the limited vocabulary, the program shows a warning to try again or to skip the turn.



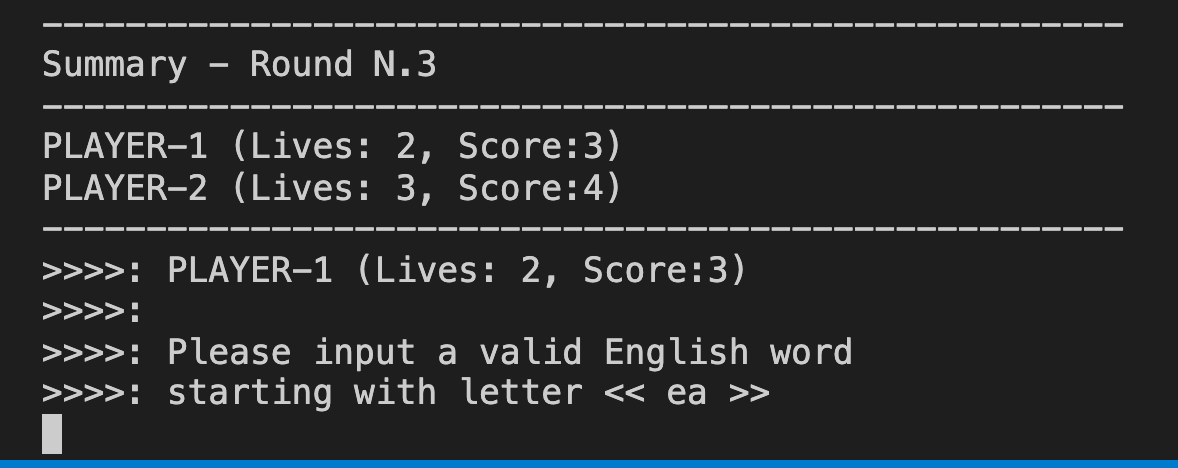
1. Player-1 types “near” which is a valid word. The passes the turn



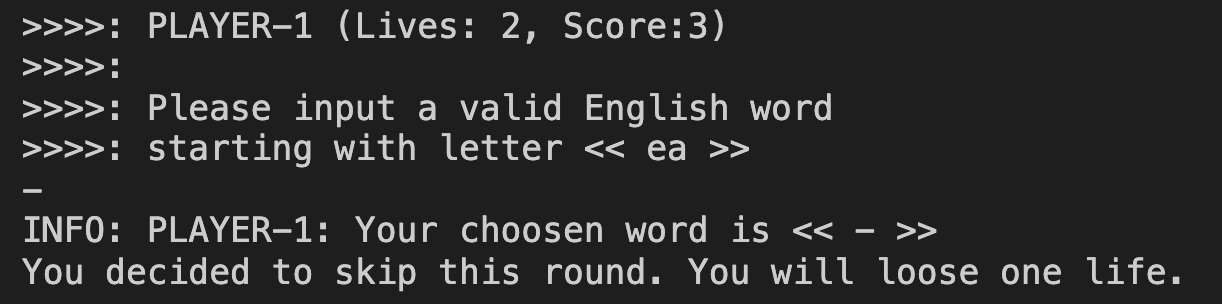
1. Is now Player-2 turn. The program asks to type a word which starts with the letters “ar”. Player-2 types “area” which is a valid word. Passes the turn.



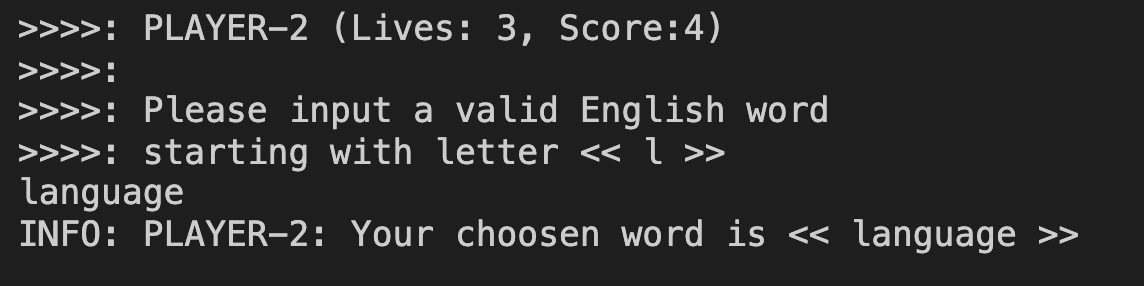
1. The program shows the Round 3 status summary. Now is the turn for Player-1 that is asked to type a word which starts with the letters “ea”.



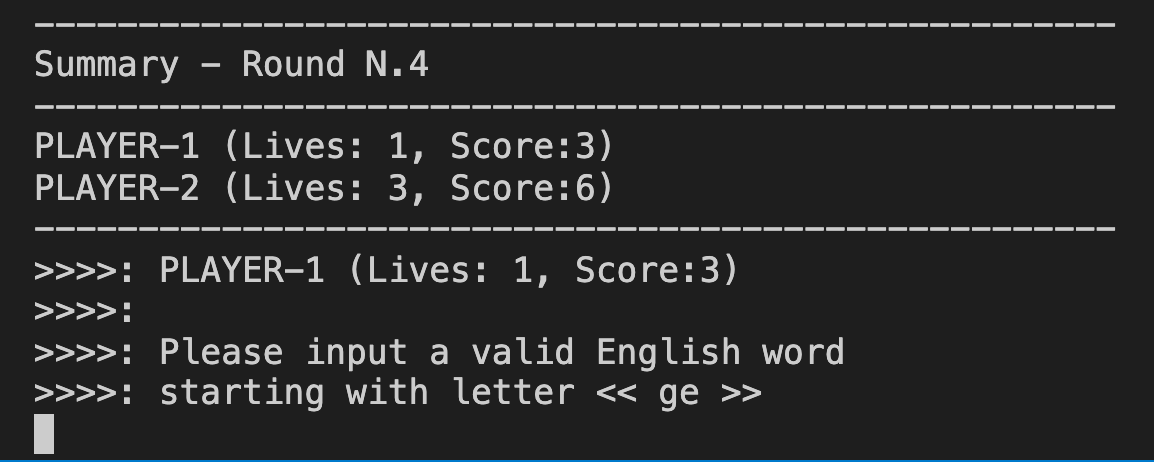
1. Player-1 decides to skip the turn and types “-”. Player-1 loses 1 life.



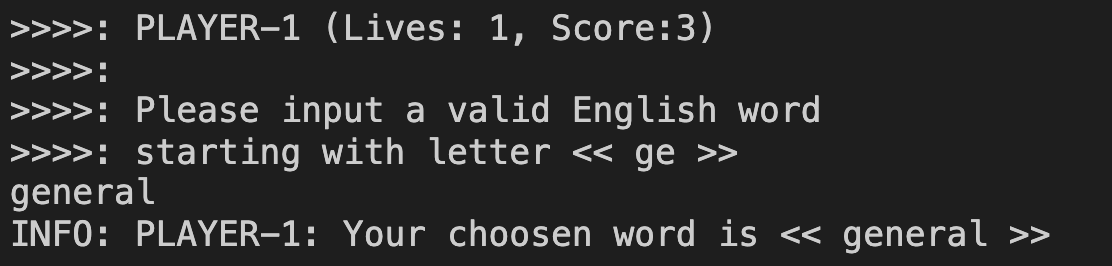
1. Is now Player-2 turn. The program asks to type a word which starts with the letter “l”. Player-2 types “language” which is a valid word.



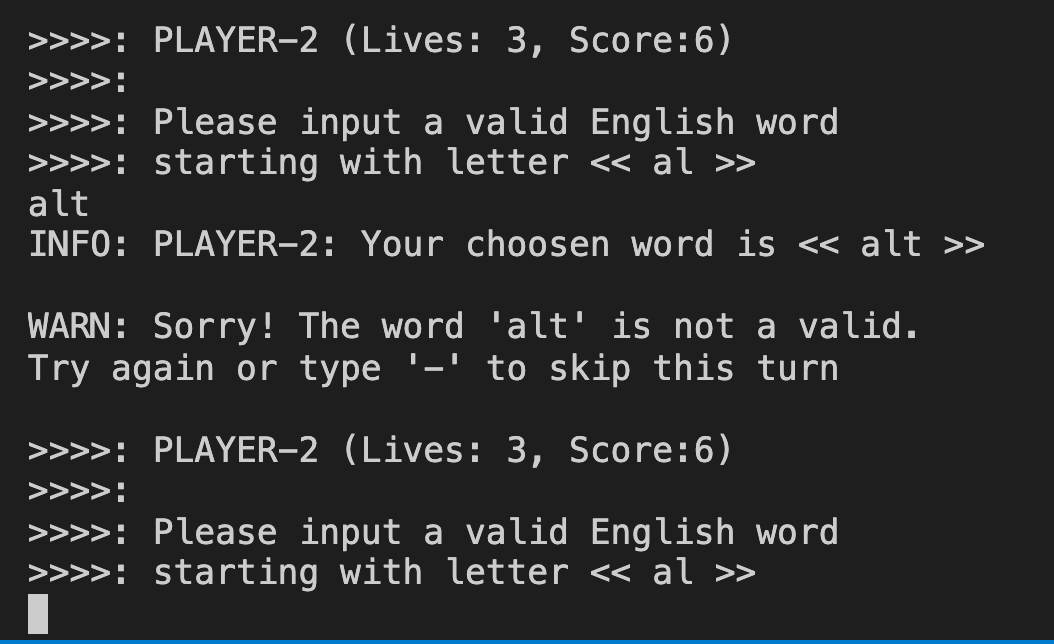
1. The program shows a summary status for round 4. Is now the turn for Player-1 who is asked to type a word which starts with the letters “ge”



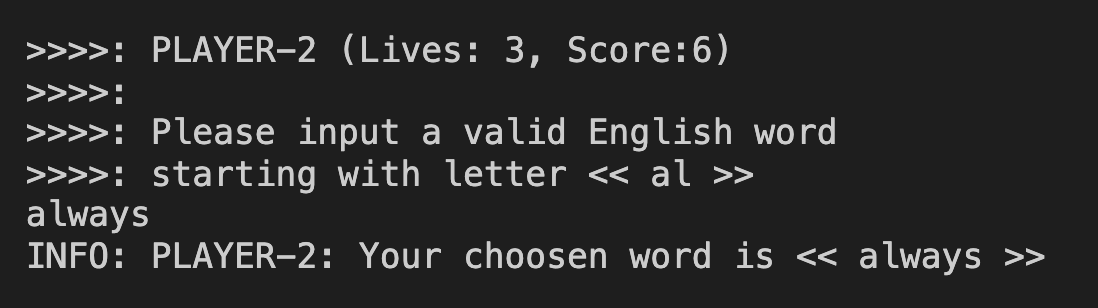
1. Player-1 types “general” which is a valid word, passes the turn



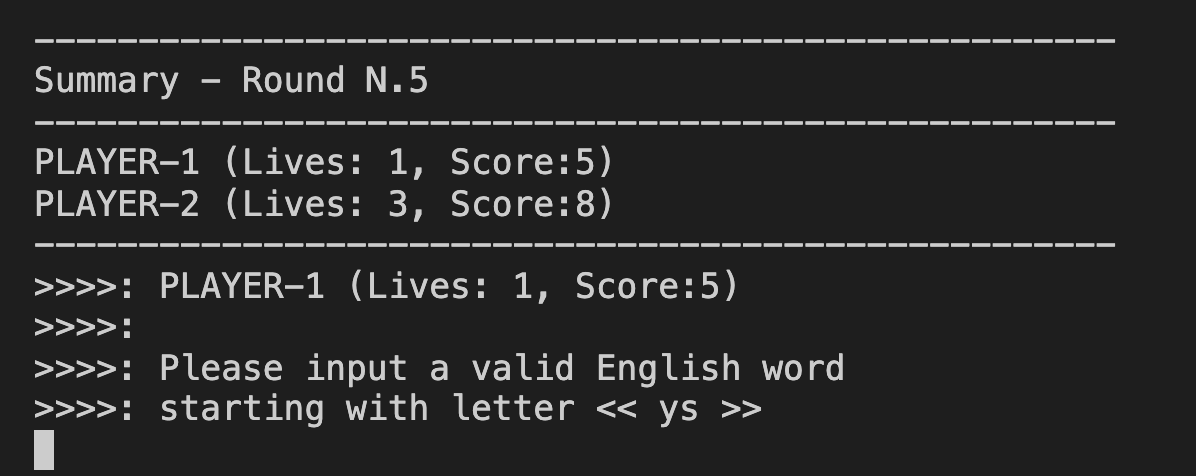
1. Player-2 starts the next turn, the word to guess must start with the letters “al”. Player-2 type “alt” which is a not valid word. The program warns to choose another word or to skip the turn typing “-”.



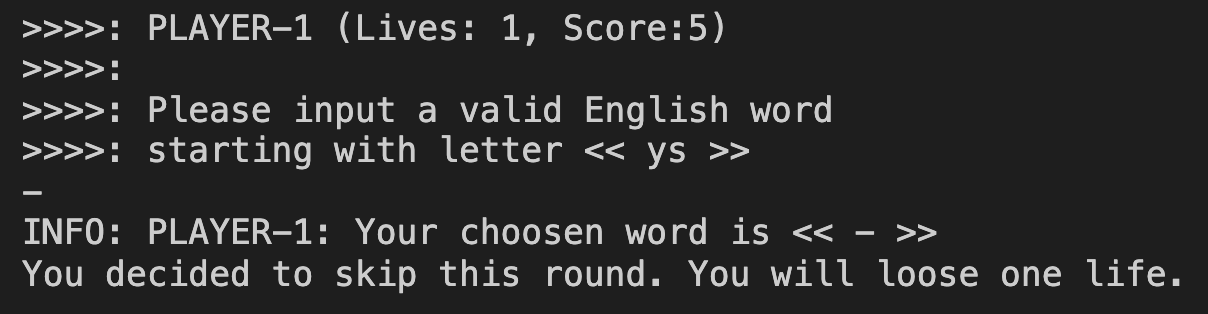
1. Player-2 types “always“” which is a valid word and passes the turn.



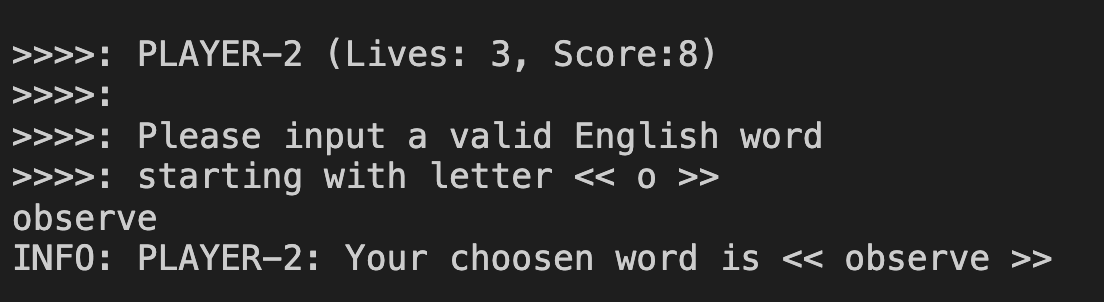
1. The program shows a summary of just ended round 5. Player-1 is the next to play. The program asks to type a word which starts with the letters “ys”.



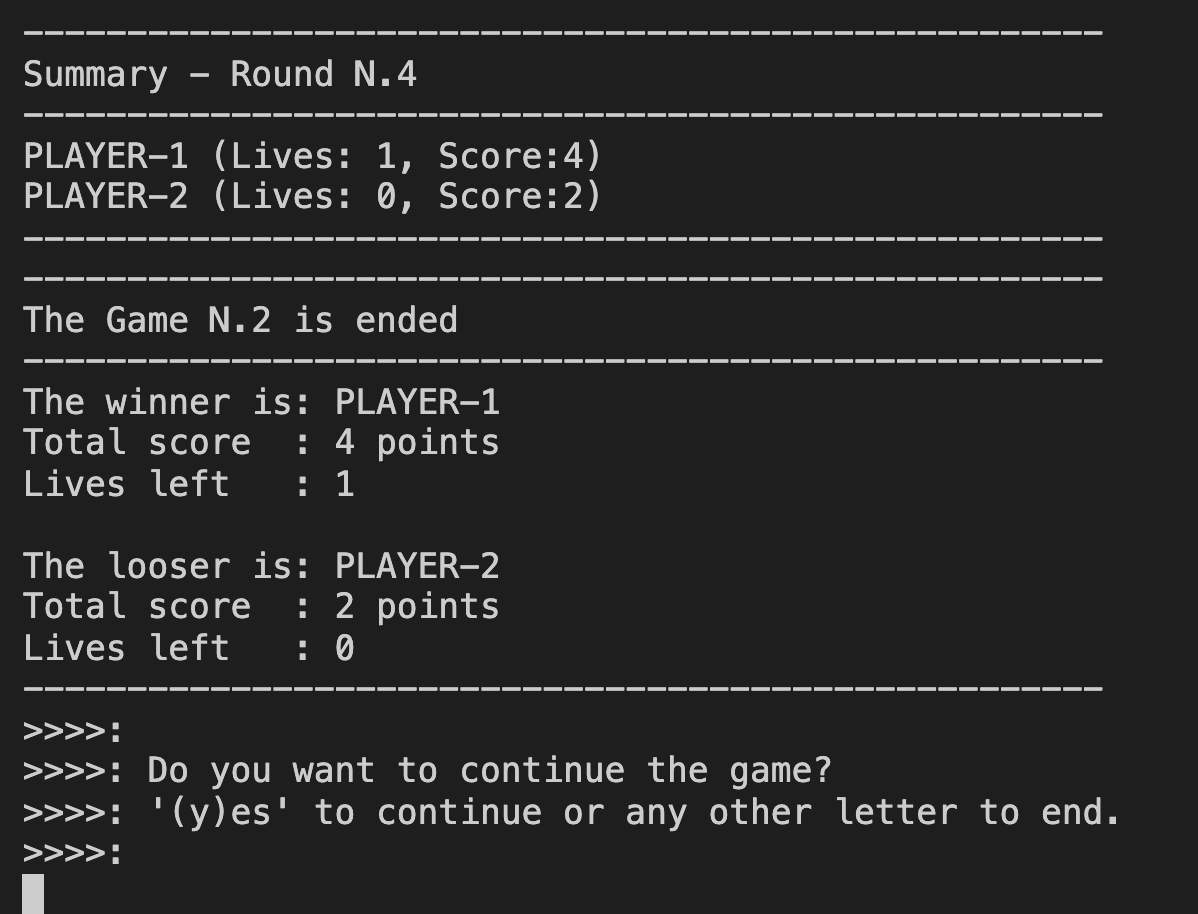
1. Player-1 decides to skip the turn and type “-”. Player-1 loses a life



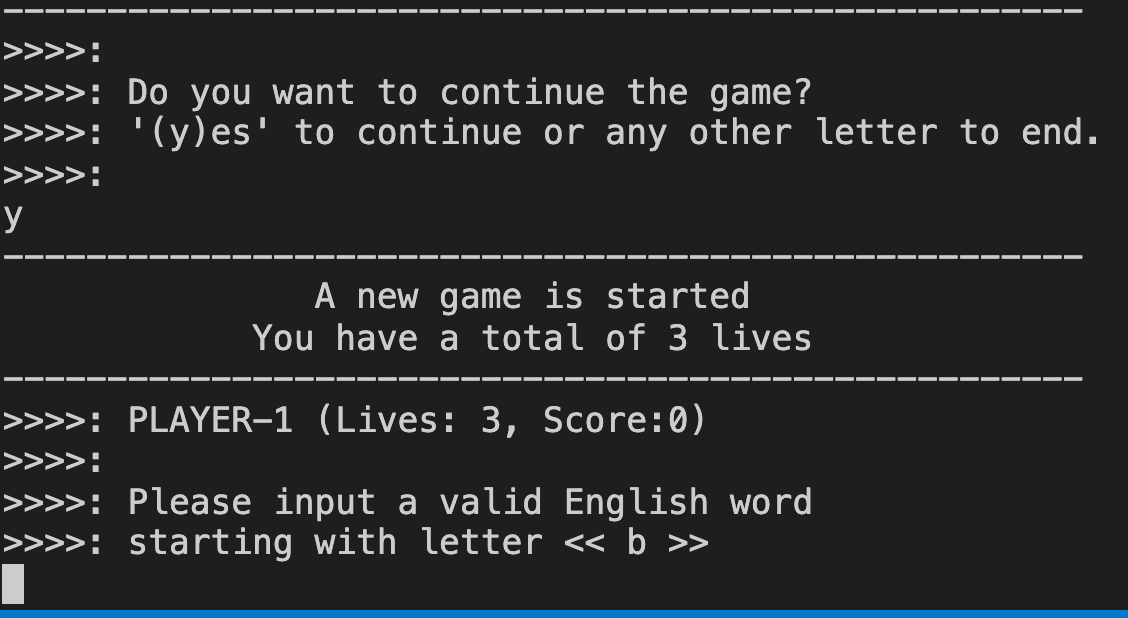
1. Is now Player-2 turn who has to guess a word which starts with the letter “o”. Player-2 types “observe” which is a valid word and passes the turn



1. The program shows a summary of the just ended round 4.   
   The program checks if any player has no more life left, than it ends the game.  
   The program displays who is the winner and who is the loser.  
   The program asks if the player wants to play another game, if so the player has to type either “yes” or “y”, if not the player can type any other letter to finish to play.



1. The player chooses to play one more game typing “y”. A new game starts



1. At the end the second game the player does want to play anymore, then type anything to end the game. The program shows a greeting message.  
   The game session is concluded



**Full testing log**

Please check the attached file ./docs/game-log.log