Software Design For Wheel of Fortune

1. A user may choose to either create a new player or log in

To realize this requirement, I added to the design a class **user** and a class **player**. 1 user can create 1 player.

2. After logging in, the application shall allow players to (1) create a puzzle phrase, (2) solve a random puzzle, (3) create a tournament, (4) join or continue a tournament, and (5) view the puzzle statistics.

To realize this requirement, I added to the **player** class with operations:**create_a_puzzle_phrase();solve_a_random_puzzle();create_a_tournament();join_or_continue_a_tournament();view_the_puzzle_statistics()**

3. The application shall maintain an underlying database to save persistent information across runs (e.g., players, puzzles, statistics, tournament information).

I did not consider this requirement because the database will only persist data already present in the design.

4. When creating a new player, a user will:Enter the player's first name. Enter the player's last name. Enter the player's desired username. Enter the player's email. Submit the information. Either receive a confirmation that the information is saved and return to the menu or receive an error if the username is already taken on that device and be offered the option to pick a different username. A player cannot be edited or deleted after a successful save.

To realize this requirement, I added to the **player** class with attributes:**first_name; last_name; desired_username; email; user_name_checked** here, I can use boolean variable **user_name_checked** to know whether the username is already taken or not. And I also add operation **submit()** to save the information of player.

5. To create a puzzle, the player will: Enter a phrase. Enter the maximum number of allowed wrong guesses a user can make before losing the game, between 0 and 10. Save and view the returned unique identifier for the puzzle. The puzzle may not be further edited after this point.

To realize this requirement, I added to the **puzzle** class with attributes:**phrase; maximum_number_of_allowed_wrong_guesses** , and with operations: **save(); view()**. 1 player can create many puzzles.

6.1 When a player starts solving a puzzle, whether selected randomly or belonging to a tournament.

To realize this requirement, I added to the **puzzle_game** class and **tournament**. Also I add association between **player** and **tournament** as select, I add association between **player** and **puzzle_game** as select_randomly.

6.2 the game will: Display the puzzle phrase, where (1) all non-alphabetic characters (e.g., numbers or punctuation) are shown, and (2) regular letters are replaced by blanks. The game should also display a list of all letters not yet

chosen, the total prize, with an initial value of $0, and the remaining number of allowed wrong guesses, initialized to the maximum number of allowed wrong guesses chosen by the puzzle creator (see above).

To realize this requirement, I added to the **puzzle_game** class with attributes:**total_prize;remaining_number_of_allowed_wrong_guesses**, with operation: **display()**. **display()** will return the above requirements such as the puzzle phrase, the list of all letters not yet chosen and so on.

6.3 Allow the player to choose, at every turn, whether to guess a consonant, buy a vowel, or solve the puzzle.

To realize this requirement, I added the **consonant** , **vowel** class with association relationship guess and buy. I also add **choose()** operation and **turn** attribute in class **puzzle_game**

6.3.1 Guessing a consonant will show the player a randomly chosen prize value that is a multiple of $100 and is between $100 and $1000. If the guess is correct (i.e., the consonant is in the puzzle), all the occurrences of the consonant in the puzzle will be revealed, and the total prize will be increased by the prize value times the number of such occurrences.

To realize this requirement, I added the **consonant** class with attributes **prize value, total prize, occurrences** and with operation **reveal_occurrences(), increase_total_prize()**

6.3.2 Buying a vowel will cost $300 of the player's total prize and will result in revealing all instances of that vowel in the puzzle.

To realize this requirement, I added the **vowel** class with attributes **prize value, total prize** and with operation **reveal_occurrences()**

6.3.3 If a vowel or a consonant are guessed incorrectly (i.e., the guessed letter is not present in the puzzle), the remaining number of allowed wrong guesses is decremented. If the number goes below zero, the player gets a prize of $0 for that puzzle, and the game ends.

To realize this requirement, I added the **consonant** and **vowel** class with attributes **guess_result** to check whether the result of guessing is correct or wrong. I also add operation **end()** to class **puzzle_game**.

6.3.4 If a player selects to solve the puzzle and is successful, he/she will score $1000 for each letter not yet revealed, and his/her total prize will be recorded and associated to that puzzle and player. Conversely, if a player tries to solve the puzzle and is unsuccessful, he/she gets a prize of $0 for that puzzle, and the game ends.

To realize this requirement, I added the **consonant** and **vowel** class with operations **solve_successfully()** and **solve_unsuccessfully()**

7. If a player interrupts a puzzle (e.g., by explicitly choosing to exit the game while solving a puzzle), the game must give the player the option to continue. If the player confirms that he/she wants to exit, he/she gets a prize of $0 for that puzzle, and the game ends.

To realize this requirement, I added the association relationship interrupts between class **player** and **puzzle_game**. I also add operation **interrupt()** to class **player**. I also add operations **continue(), exit()** to class **puzzle_game**.

8. When a player selects to solve a random puzzle, the game will not chose puzzles he/she has created or already successfully/unsuccessfully played.

To realize this requirement, I added the association relationship select_randomly between class **player** and **puzzle_game**.

9. When a player selects a tournament for which he/she has already played some of the puzzles, the game will consider these puzzles already completed and preserve the prize the player won (including $0 for puzzles the player quit or did not successfully solve).

To realize this requirement, I added the class **puzzle_game** with attribute **played** to check whether these puzzles already completed or not.

10. To create a tournament, a player will: Select 1 to 5 puzzles from a list of puzzles that they have either created or

already played. Enter a name for the tournament. Either receive a confirmation that the tournament has been created and return to the menu or receive an error if the tournament name is already taken and be offered the option to pick a different name. At this point, the tournament will be available for others to join.

To realize this requirement, I added the class **tournament** with attribute **name, check_name_taken** and with operations **select_puzzles()**

11.To play a tournament, a player can select whether to join a new tournament or continue a tournament he/she has already joined.

To realize this requirement, I added the class **tournament** with operation **join_a_new_tournament(), continue_a_tournament()**

11.1 If the player opts for joining a new tournament, the game will show the player a list of tournaments that are currently available for him/her to join (i.e., all puzzles not created and not yet played by the player all tournaments that contain (1) no puzzles created by the player and (2) at least one puzzle not yet played by the player). When the player chooses a tournament in the list, the game will display the first puzzle in the tournament.

To realize this requirement, I added the class **tournament** with attribute **available_tournaments**

11.2 If the player opts for continuing a tournament they have already joined, the game will show the player a list of tournaments they are currently playing that still have puzzles not completed by the player. When the player chooses a tournament in the list, the game will display the first unsolved puzzle in that tournament.

To realize this requirement, I added the class **tournament** with attribute **check_joined** to check whether this tournament is used or not

11.3 After a player completes the last puzzle in a tournament, the tournament ends (for that player), and the game stores the overall tournament prize of the player, which is the sum of the player's total prizes in all the puzzles in the tournament.

To realize this requirement, I added the class **tournament** with operations **end()** and with attribute **overall_tournament_prize**

12. When a player opts to view the puzzle statistics, the game will show four pieces of information:

To realize this requirement, I added the accociation relationship view puzzle statistics between class **player** and **puzzle_game**. Besides, I also add operation **view_the_puzzle_statistics()** to class **player**.

12.1 The list of puzzles completed by that player with, for each puzzle, the prize the player won (including $0 for puzzles he/she quit or did not successfully solve).

To realize this requirement, I added the class **puzzle game** with attribute **list_of_puzzles_completed** which is a string array to store the list of puzzles completed by that player with and **map<string,money>player_prize** which is hashtable to store the prize the player won in each puzzle.

12.2 The list of tournaments completed by that player with, for each tournament, the prize the player won.

To realize this requirement, I added the class **tournament** with attribute **list_of_tournaments_completed** which is a string array to store the list of tournaments completed by that player with and **map<string,money>tournament_prize** which is hashtable to store the prize the player won in each tournament.

12.3 The complete list of puzzles with, for each puzzle, (1) the number of players who played it and (2) the top prize won by a player for that puzzle, together with the username of that player.

To realize this requirement, I added the class **puzzle game** with attribute **number_of_players**

12.4 The complete list of tournaments with, for each tournament, (1) the number of players who completed the tournament and (2) the top prize won by a player for that tournament, together with the username of that player.

To realize this requirement, I added the class **tournament** with attribute **number_of_players**

13.The User Interface (UI) shall be intuitive and responsive.

This requirement has nothing to do with UML graph.

14.The performance of the game should be such that players does not experience any considerable lag between their actions and the response of the game.

This requirement has nothing to do with UML graph.