10.014 Computational Thinking for Design 1D Project
PET Type Racing Game
Group 3B

	Team Member	Student ID
1	Brayden Tang Jia Jun	1005897
2	Goh Jian De	1005912
3	Michelle Chrisalyn Djunaidi	1006240
4	Darren Chan Yu Hao	1006340
5	Lim Zhi Xian, Jordan	1006024

Description

Scenario:

Personal English Trainer (PET) type racing game is a multiplayer text based game where two or more players battle it out to test their knowledge and range of English vocabulary. Designed for students in mind, this game aims to expand and improve on their vocabulary, and learn new English words.

With the widespread prevalence of technology, such as mobile phones and computers, children mainly consume entertainment from such outlets, neglecting print media such as books and newspapers. This has undoubtedly caused a gap in terms of the appreciation of the English language, and as such, our game hopes to tackle this issue.

We propose this game as a tool to gamify the teaching and learning of vocabulary, to pique students' interest in the beauty of the English language. This serves as a fun way to test student's on their knowledge. By changing the dictionary inside the database, the game can be customized to teach or learn different languages.

Description of the game:

This game is a multiplayer text based game. When the game starts, the first player will choose a prefix for the game. Each player is allocated 20 seconds as their health, each turn will deduct their health based on the time spent on their turn.

Players will use the chosen prefix to type the word, the game will record the time spent on their turn and deduct their health before proceeding to the next player's turn.

When players enter an invalid word (i.e. wrong spelling, repetition), the game will prompt him to try again. The player loses if he runs out of time, the game will continue until there is 1 player remaining.

Benefits/Educational aspect of the game:

The primary educational benefit from this game is the expansion of vocabulary range and improving the verbal fluency of students. Through this game, students are exposed to a large range of English words in the English Dictionary (around 354,984 words are included in our word list). Through constant repetition by repeatedly playing the game, students will be able to learn more English words and increase their memory retention of these words.

With the spectator function, teachers can observe the progress of their students and identify which words and prefixes the students struggled the most with. As a result, teachers can adjust their teaching methods to better accommodate the students, further personalizing and enhancing their learning experience.

To further improve the effectiveness and learning experience of this game, sentences can also be brought into the game such that students not only gain exposure to the word by itself, they would also learn the meaning of these words and how to use them contextually in proper sentences. The addition of a time element in this game also increases the ability of students to think on their feet as they would have to formulate specific words with accuracy under a short amount of time.

Key Features

1. Multiplayer function with virtually no limits on the number of players through the usage of firebase database!
2. Time based racing-esque feature that gets your blood pumping!
3. Spectator feature for teachers, friends and family to watch you battle it out!

Game Flow

When players join the game, they will be greeted with a lobby screen.

The first player, under the teacher's instructions, will choose the prefix for this round. All players must input English words starting with the chosen prefix. For example, if the first player chooses the prefix 'a', words that start with 'a' must be keyed in. Prefixes don't need to be a single letter; inputs like 'ab' or 'pass' are also accepted. This could be useful especially if teachers are teaching different forms of a verb, like passes/passed/passing.

After choosing the prefix, the first player will then choose how many players should join the game. The game will start once there are enough players.

To make the game more competitive and challenging, there will be a timer for each player to type the correct words. If no correct words are typed in the *given time frame*, the player will lose. Players are allowed to type in as many words as they want in the time frame. They are encouraged to try again if they misspell a word. If the player has typed a word that has been used by other players, the player will be prompted to try again.

When there is only one player left, a message would be displayed on screen, declaring that player as the winner.

Our game utilizes the firebase to create a seamless online multiplayer experience. As such, students are able to play the game with their own laptop anywhere, anytime. With an in-built

spectator function, spectators such as teachers can monitor the flow of the match and edit the database as needed. Want students to learn another language instead of English? Just edit the dictionary. Want students to learn past tense forms of words? Only put past tense words in the dictionary. Our game is flexible and adaptable to the needs of the classroom.

Documentation

In this section, the code base will be documented and explained in detail. It is split into 5 main sections:

1. Libraries
2. Functions
3. Setup
4. Main
5. Firebase Database

In each section, in depth explanation what each part of the code will do will be stated.

Libraries

```
#================================================================================
# Import Libraries
#================================================================================
from libdw import pyrebase #For the  multiplayer aspect
import os # For clearing screen if needed
import time # For timer based actions
import msvcrt # For non-paused user inputs
#================================================================================
```

In this project, 3 python standard libraries and libdw library and pyrebase (a python firebase wrapper) are used

In this section, the purpose of each library will be described.

1. Libdw | Pyrebase:

   Libdw is a package created by Prof Oka Kurniawan from Singapore University of Technology and Design (SUTD-ISTD) for varying purposes. In Computational Design Thinking course for Freshmores, pyrebase library found packaged in libdw is taught.

   Pyrebase is a simple python wrapper for the Firebase API by James Childs-Maidment and can be found on Github : https://github.com/thisbejim/Pyrebase

   In this program, the main purpose of this library is to enable the client side code to read and write to a designated Firebase database to enable turn-based online multiplayer between multiple computers.

2. OS:
   The OS library is a standard python library that provides access to the user operating system dependent functionality.
   In this program, only the function os.system(“cls”) is used. This function is called to clear the terminal screen.

3. Time:

   The time library is a standard python library that provides a timer functionality.

   In this program, two functions from this library are called time.sleep(secs) and time.time().

   In this program, time.sleep(secs) is used to pause the program when it wishes to display a message for an extended period, for instance, when a player loses, it will print a

statement to inform the player that he/she has lost and after a set period of time, the game will quit.

time.time() is used in this program as a timer. time.time() returns a <float> number commonly known as Epoch time. In most systems, this time is calculated by the number of seconds that have passed since 00:00:00 UTC on 1 January 1970. By calling this function right before and right after a player turn, the time the player took can be calculated.

4. msvcrt (Microsoft Visual C/C++ Runtime Library):

   The msvcrt is a standard python library that provides access to some windows functionality.

   In this program, the main purpose for calling this library is to use functions that are related to keypress detection events. This is useful in situations where the program cannot afford to pause just to wait for a player's input.

   The downside of this library is that it limits this program to window users only.

Functions

get_dictionary()

Returns the english dictionary from the database. This value is returned as a <list>.

```
def get_dictionary():
   key = "Dictionary"
   node = db.child(key).get(user['idToken'])
   english_dictionary = node.val()
   return english_dictionary.split(' ')
```

player_input_word(prefix <string>, english_dictionary <list>, wordlogger <list>)

Input is stored in player_word as a <string>. Append player_word into wordlogger if player_word is inside the english_dictionary and it isn't inside wordlogger.
Set database's row wordlogger with the new wordlogger.

```
def player_input_word(prefix,english_dictionary,wordlogger):
   print("Key in your word")
   player_word = (prefix + input("{}".format(prefix)) ).lower()
   if player_word in english_dictionary and player_word not in wordlogger:
       set_wordlogger(wordlogger,player_word) # adds player_word into wordlogger
       return True
   else:
       return False
```

player_input_prefix()

Input is stored in prefix <string>.
Set database's row prefix with prefix <string>.

```
def player_input_prefix():
   prefix = input("Key in this round's prefix: ")
   prefix = prefix.upper()
   key = "prefix"
   db.child(key).set(prefix, user['idToken'])
   return prefix
```

get_prefix()

Returns the prefix from the database. This value is returned as a <string>.

```
def get_prefix():
   key = "prefix"
   node = db.child(key).get(user['idToken'])
   prefix = node.val()
   return prefix
```

choose_player()

User enters input <string> . This returns an <integer>
The code will print when a new player joins the lobby.

1. If the player enters ‘y’, it gets player_count <integer> from the database’s row ‘player_count’, it sets the database’s row ‘player_count’ with the new player_count.
2. If the player enters ‘n’, the code will raise TypeError and the program will close after 2 seconds.
3. If the player enters ‘r’, the code will reset the database.

```
def choose_player(): #player_number_initialization (brayden)
   done = False
   player_count = 0
   print("{} player(s) waiting in the lobby. Press 'y' to join, 'n' to quit! (Press 'r' to reset if game is
stuck)".format(player_count))
   while not done:
       previous_player_count = player_count
       key = "player_count"
       node = db.child(key).get(user['idToken'])
       player_count = node.val()
       if(previous_player_count != player_count):
           os.system('cls')
           print("{} player(s) waiting in the lobby. Press 'y' to join, 'n' to quit! (Press 'r' to reset if
game is stuck)".format(player_count))
       if msvcrt.kbhit():
           player_input = msvcrt.getch().decode('ASCII').lower()
           if player_input == 'y':
               done = True
               player_count += 1
               print(player_count)
               db.child(key).set(player_count, user['idToken'])
               print("Welcome Player {}!".format(player_count))
               return player_count
           if player_input == 'n': # New players press 'n' will exit
               print("You have quit")
               time.sleep(2)
               raise TypeError
           if player_input == 'r':
               reset_stage(0)
```

get_current_player()

Returns the current player node from the database. This value is returned as a <string>.

```
def get_current_player():
   key = "current_player"
   node = db.child(key).get(user['idToken'])
   current_player = node.val()
   return current_player
```

set_current_player(current_player <int>)

Set the database’s ‘current_player’ row with the parameter <string>

```
def set_current_player(current_player):
   key = "current_player"
   db.child(key).set(current_player, user['idToken'])
```

get_player_count()

Returns the total number of registered players (including spectators) in the game. This value is a <integer>

```
def get_player_count():
   key = "player_count"
   node = db.child(key).get(user['idToken'])
   player_count = node.val()
   return player_count
```

get_max_number_of_players()

Returns the maximum number of players allowed in the game. This value is an <integer>.

```
def get_max_number_of_players():
   key = "max_number_of_players"
   node = db.child(key).get(user['idToken'])
   number_of_players = node.val()
   return number_of_players
```

set_max_number_of_players()

Function requires input <int> which is stored in number_of_players <int>.
Set the database's row of 'max_number_of_players' <int>, 'Health' <dict>, 'players_alive' <list>.

```
def set_max_number_of_players():
   number_of_players = int(input("Set number of players in this round: "))
   key = "max_number_of_players"
   players_alive = []
   for player_name in range(1,number_of_players+1):
       add_to_dictionary = {str(player_name):20}
       db.child('Health').update(add_to_dictionary, user['idToken'])
       players_alive.append(str(player_name))
   db.child('players_alive').set(players_alive,user['idToken'])
   db.child(key).set(number_of_players, user['idToken'])
```

set_check_set_up_status(key <string>)

Set the database's row of 'set_up_status' <boolean>.

1. If the key is "Start Game", the function will set it as True.
2. Otherwise it will leave it as False (by default) .

```
def set_check_set_up_status(key):
   if(key == "Start Game"):
```

```
        key = "set_up_status"
        db.child(key).set(True, user['idToken'])
```

get_check_set_up_status()

Returns the set-up status from the database. This value is returned as a <boolean>

```
def get_check_set_up_status():
    key = "set_up_status"
    node = db.child(key).get(user['idToken'])
    set_up_status = node.val()
    return set_up_status
```

get_players_alive()

Returns the players alive from the database. This value is returned as a <list>

```
def get_players_alive():
    key = "players_alive"
    node = db.child(key).get(user['idToken'])
    get_players_alive = node.val()
    return get_players_alive
```

set_players_alive(players_alive <list>)

Set the database's row of 'players_alive' with the parameter <list>

```
def set_players_alive(players_alive):
    db.child('players_alive').set(players_alive,user['idToken'])
```

get_wordlogger()

Returns the word logger from the database. The word logger consists of words that have been keyed by the players. This value is returned as a <list>

```
def get_wordlogger():
    key = "wordlogger"
    node = db.child(key).get(user['idToken'])
    return node.val()
```

set_wordlogger(wordlogger <list> , word <string>)

Function appends the new word <string> into the wordlogger <list>, it will then set the database's row 'wordlogger' with the new 'wordlogger' list

```
def set_wordlogger(wordlogger,word):
    wordlogger.append(word)
```

```
   db.child('wordlogger').set(wordlogger,user['idToken'])
```

get_health()

Returns the health of all players. This value is returned as a <dict>

```
def get_health():
   key = "Health"
   node = db.child(key).get(user['idToken'])
   return node.val()
```

set_health()

Set the database's row 'Health' with the parameter health <dict>

```
def set_health(health):
   db.child('Health').set(health,user['idToken'])
```

reset_stage(player_count <int>)

Resets the database's row for 'player_count', 'current_player', 'set_up_status', 'max_number_of_players', 'prefix' to their designated pre-game conditions.

In the case for 'player_count':

1. If player number is 1, it will reset the player_count
2. Otherwise, it will set the player_count as 0. This will be used to reset the game when it ends

```
def reset_stage(player_number):
   key = "player_count"
   if(player_number == 1):
       db.child(key).set(1, user['idToken'])
   else:
       db.child(key).set(0, user['idToken'])
   key = "current_player"
   db.child(key).set('1', user['idToken'])
   key = "set_up_status"
   db.child(key).set(False, user['idToken'])
   key = "max_number_of_players"
   db.child(key).set(999, user['idToken'])
   key = "prefix"
   db.child(key).set(" ", user['idToken'])
   db.child('wordlogger').set([''], user['idToken'])
```

Setup

When the code is started up, a few functions will be called to get the stage ready for the game.

1. get_dictionary()

   This function is called to obtain the dictionary from the firebase for the program to reference in the game

2. get_player_count() and get_max_number_of_players()

   Count how many players are currently registered in the game and also what is the max number of players allowable for this round. (Note: The start state for max_number_of_players, if not yet set by any player will be 999, this is to allow for spectators such as teachers to monitor the game in real time)

```
#=============================================================================================================
# Startup and Global Variables
#=============================================================================================================

# Get a string of english words using the get_dictionary method and setting it to the string variable
english_dictionary
english_dictionary = get_dictionary()

# Count how many players are currently registered in the game and setting it to the int variable player_count
player_count = get_player_count()

#Check what is the max number of player allowed for this round and setting it to the int variable
max_player_count
max_player_count = get_max_number_of_players()
```

With player_count and max_player_count variable values properly assigned, the program compares the two values using an if statement to see if the lobby is already full.

The if statement compares more than or equal to because, at this initial comparison, the user is not yet a registered player, meaning he/she is not yet counted in player_count.

If the lobby is full, it will enter into the if statement code block, here we will print an initial statement informing the player that the lobby is full.

We then give the player an option to reset the lobby by typing in admin. This action will reset the database, thereby clearing the lobby.

Once the player has reseted the lobby, he/she can then join a new game.

```
if(player_count >= max_player_count):
```

```
    print("Sorry the lobby is full for this round!")

    while(player_count >= max_player_count):
        check_admin = input("reset the lobby by typing admin: ").lower()

        if(check_admin == "admin"):

            reset_stage(0)
            player_count = get_player_count()
            max_player_count = get_max_number_of_players()
```

If the player is allowed to join the game (e.g game has yet to start and there are still slots), the player will be sent to a lobby via the choose_player() function. This function will prompt the player if he/she wishes to join the game. Documentation for choose_player().

By doing this, the user is now registered as a player by the firebase database. The program then gets a variable that will indicate if the game is ready to be played. If it is, set_up_status will be set to True, if it is not, it will be set to False. Documentation for get_check_set_up_status().

```
player_number = choose_player() #Allow player to join the game by selecting their player number

set_up_status = get_check_set_up_status() #Check if the game is ready to be played!
```

If the player is the first player of the lobby, he/she will be designated as the host of the round. The host player will first reset the game lobby so as to clear any previous game's variables, if they are not yet cleared by calling the reset_stage() function.

He/she will then be prompted to set a prefix and maximum number of players for this round of the game via player_input_prefix() and set_max_number_of_players()

If the player is not the host, the player will update the number of registered players in the lobby and also the max number of players and compare if the lobby is filled or not, again. This is because, while he/she is deciding to join the game and waiting at the idle screen, the game might have filled up with enough players already.

If everything is good, the game will check and wait for player 1 to finish setting up before proceeding to start the game.

```
if (player_number == 1):

    # Reset Stage
    reset_stage(player_number)

    # Start new stage
    prefix = player_input_prefix()
    number_of_players = set_max_number_of_players()
```

```
    set_up_status = set_check_set_up_status("Start Game")

else: #This is for if you are not player 1

    player_count = get_player_count()
    max_player_count = get_max_number_of_players()

    # If it turns out you joined too late, it will tell you the lobby is full
    if(player_count > max_player_count): #This one is less than only
        print("Sorry the lobby is full for this round!")
        time.sleep(5)
        raise TypeError

    # This part waits for the game to be done setting up by player 1
    if(set_up_status != True):
        print("Waiting for Player 1 to finish setting up!")
        while(set_up_status != True):
            set_up_status = get_check_set_up_status()
```

To finish the set-up, all players will now be allocated the predefined prefix set by player 1 via the get_prefix() function.

Once done, everybody will get the current player's turn by calling the get_current_player() function. This will set all players to the same turn.

```
    #Once game is set up, you will get the prefix set up by player 1 for this round
    prefix = get_prefix()

# This will set the current turn universally across all registered player to the same player
current_player_turn = get_current_player()
```

Main game

While loop will iterate after every player's turn.
max_player_count is set by the game administrator, this value represents the maximum number of registered players allowed in the game.
The code will constantly check if the player_count is equal to the max_player_count, the game will only proceed once the number of registered players have filled up the lobby.

```
while True:
   players_alive = get_players_alive()
   if(player_count < max_player_count):
       print("Waiting for other players to join!")
       while(player_count < max_player_count):
           player_count = get_player_count()
       print("The game has started")
```

The code below describes how the game interacts with the player when it isn't their turn.
In the while loop, it will call the database with get_current_player() to check which player's turn. If it isn't the player's turn, it will print the current player's turn.
The code will call the database with get_players_alive() and store it inside the players_alive list, if there is only 1 player then the console will return a feedback that the game has concluded and the player has lost.

```
   if (player_number != current_player_turn):
       print("It is now player {} turn".format(current_player_turn))
       while (player_number != int(current_player_turn)): # isn't your turn
           previous_player_turn = current_player_turn      # This will be used to check if there has been any
change in player
           current_player_turn = get_current_player()      # Keep querying current player's turn
           players_alive = get_players_alive()             # Check who is alive
           wordlogger = get_wordlogger()

           if(current_player_turn != previous_player_turn):

               if(str(current_player_turn) in players_alive):
                   print("It is now player {} turn".format(current_player_turn))
                   print("These are the current words that have been entered: {}".format(wordlogger[1:]))

           if len(players_alive) == 1:     # When overall game concludes (aka not just when you lose), this is
the what will be shown to all the losers
               if (str(player_number) not in players_alive):

                   set_current_player(int(players_alive[0]))
                   input('The game is over, try harder next time!')
                   raise TypeError
```

The code below describes when it has reached the player's turn.
The game will check if the player is inside players_alive list, the game will skip the player's turn if he is no longer alive. The client will update the database's row 'current_player_turn'.

If there is only 1 person in the players_alive list and it is the player's turn, he is the champion

and the code will reset the database before closing the programme with TypeError.

```
while(player_number == int(current_player_turn)):  # If it is your turn, player_number is stored as an integer
       if (str(player_number) not in players_alive):
           next_player_number = player_number +1
           if next_player_number > max_player_count:
               next_player_number = 1
           set_current_player(next_player_number)
           current_player_turn = get_current_player()
           continue
       players_alive = get_players_alive()
       if(len(players_alive) == 1 ):
           if (player_number == int(players_alive[0])): # Check if the player_number is the last survivor
               reset_stage(0) # Reset game after game over
               input('Well played champ')
               raise TypeError
```

The code below will validate the player's word and eliminate the player if he has run out of health. The game uses get wordlogger() to retrieve the list of words that has been used by other players.

If the word has been logged inside word_logger or if it is not included inside the english_dictionary, then it will prompt the user to try a new word.

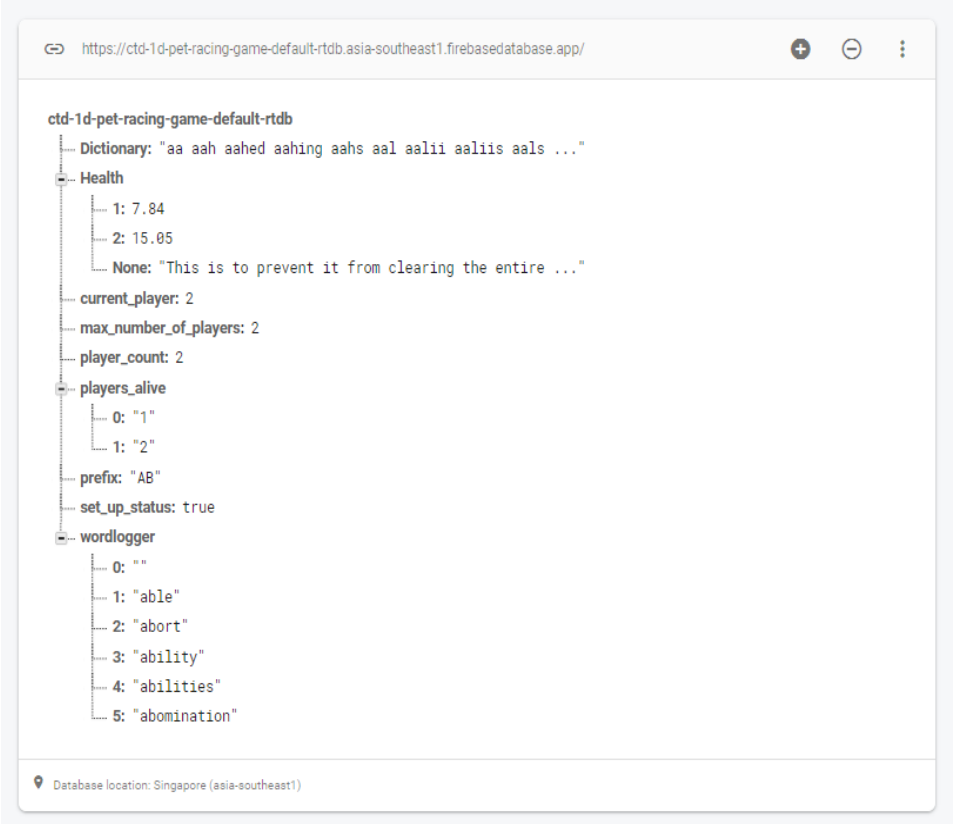
When the user has run out of health, he will be eliminated from the game. The database will remove him from the players_alive list and his turn will be subsequently skipped.

The database's row 'health' and 'players_alive' will be updated.

```
health = get_health()
       wordlogger = get_wordlogger()
       correct = False
       start = time.time()
       while(correct == False):
           correct = player_input_word(prefix,english_dictionary,wordlogger)
           if(correct == False):
               print("Not a valid word")
           time_used = time.time() - start
           if time_used > health[str(player_number)]:
               break
       health[str(player_number)] = round(float(health[str(player_number)]) - time_used,2)
       print("This is remaining hp: {}".format(health[str(player_number)]))
       if health[str(player_number)] > 0:
           set_health(health)
       else:
           del health[str(player_number)]
           players_alive.remove(str(player_number))
           set_health(health)
           set_players_alive(players_alive)
           print('You have lost, better luck next time')
       next_player_number = player_number + 1
       if next_player_number > max_player_count:
           next_player_number = 1
       set_current_player(next_player_number)
       current_player_turn = get_current_player()
```

Database

In this section, the different child nodes of the database will be described



Dictionary <string>
Stores the english dictionary. This will be referenced in the game, the code will use the dictionary to verify if the user input is valid.

Health <Dictionary>
Stores the health of each player. I.e. Health[‘1’] = 7.84 represents player 1 has 7.84 health

Current_player <integer>
Stores the current player’s turn

Max_number_of_players <Integer>
Stores the maximum number of registered players allowed in the game.

Player_count <integer>
Stores the number of players (including) spectators inside the game.

Players_alive <list>

Store a list of players that are alive.

Set_up_status <Boolean>
Stores a boolean, this will help guide the program to initialise the game.

Wordlogger <list>
Store a list of words that has been keyed by the players