**LAPORAN TUGAS BESAR**

**IF1210 DASAR PEMROGRAMAN**

**TUGAS BESAR SISTEM INVENTORI BNMO**



Disusun Oleh:

**Kelas K04 – Kelompok 4**

Kenneth Ezekiel Suprantoni 16521040

M. Bharata Sri Prana Ludira H. 16521148

Melvin Kent Jonathan 16521247

Noel Christoffel Simbolon 16521355

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**TAHUN 2022**

# **HALAMAN PERNYATAAN KELOMPOK**

“Saya menyatakan bahwa saya mengerjakan tugas besar ini dengan sejujur-jujurnya, tanpa menggunakan cara yang tidak dibenarkan. Apabila di kemudian hari diketahui saya mengerjakan tugas besar ini dengan cara yang tidak jujur, saya bersedia mendapatkan konsekuensinya, yaitu mendapatkan nilai E pada mata kuliah IF1210 Dasar Pemrograman Semester 2 2020/2021.”

Kenneth Ezekiel (16521040)

M. Bharata Sri Prana Ludira H. (16521148)

Melvin Kent Jonathan (16521247)

Noel Christoffel Simbolon (16521355)

# **DAFTAR ISI**

[**HALAMAN PERNYATAAN KELOMPOK** i](#_Toc101733375)

[**DAFTAR ISI** ii](#_Toc101733376)

[**DAFTAR TABEL** iii](#_Toc101733377)

[**DAFTAR GAMBAR** iv](#_Toc101733378)

[**I.** **DESKRIPSI PERSOALAN** 1](#_Toc101733379)

[**II.** **DAFTAR PEMBAGIAN KERJA ANGGOTA KELOMPOK** 3](#_Toc101733380)

[**III.** **CHECKLIST** 5](#_Toc101733381)

[**IV.** **DESAIN COMMAND UNTUK SETIAP PRIMITIF** 6](#_Toc101733382)

[**V.** **DESAIN KAMUS DATA** 8](#_Toc101733383)

[**VI.** **DESAIN DEKOMPOSISI ALGORITMIK DAN FUNGSIONAL PROGRAM** 13](#_Toc101733384)

[**VII.** **SPESIFIKASI UNTUK SETIAP MODUL YANG DIBUAT MAIN PROGRAM** 20](#_Toc101733385)

[**VIII.** **HASIL SCREENSHOT PENGUJIAN PROGRAM BERDASARKAN FITUR-FITUR PADA SPESIFIKASI** 37](#_Toc101733386)

# **DAFTAR TABEL**

Tabel 1 Daftar Pembagian Kerja Anggota Kelompok

Tabel 2 Checklist Pengerjaan Modul

# **DAFTAR GAMBAR**

[Gambar 1 Flowchart modul 02 17](#_Toc101697188)

[Gambar 2 Flowchart modul 03 17](#_Toc101697189)

# **DESKRIPSI PERSOALAN**

Masalah yang diberikan untuk kami pecahkan dalam Tugas Besar IF1210 Dasar Pemrograman ini pada intinya adalah menemukan suatu cara untuk mengambil data dari sebuah database, memodifikasinya, lalu menyimpannya kembali kedalam database tanpa menggunakan library, alias manipulasi data tanpa library. Dimana modul-modul yang diberikan kebanyakan adalah cara-cara untuk sorting, modifikasi data, menambahkan atau mengurangi data, dan juga menyimpan data itu sendiri. Dalam background permasalahannya disebutkan bahwa kami harus memperbaiki BNMO yang rusak dibanting oleh Indra yang rugi karena *gacha.* Dalam merancang program utama untuk memecahkan permasalahan ini pun, dibutuhkan teknik programming *modular programming*, dimana sebuah program besar dapat dipecah-pecah menurut fungsionalitas-fungsionalitasnya.

**F02 – Register**

Subprogram register digunakan untuk menambahkan User ke database BNMO, lebih tepatnya ke user.csv. Subprogram ini hanya dapat diakses oleh Admin, dan pengguna yang ditambahkan menggunakan subprogram ini hanya dapat berupa User. Tidak bisa menambahkan pengguna dengan role Admin menggunakan subprogram ini. Untuk menambahkan pengguna dengan role Admin, dapat langsung mengedit database user.csv.

**F03 – Login**

Subprogram login bertugas untuk mengecek kevalidan login data yang diinput oleh pengguna. Login data tersebut berupa username dan password. Subprogram ini akan mengembalikan data bertipe boolean yang bernilai True jika username dan password benar dan terdapat pada database.

**F04 – Menambah Game ke Toko Game**

**F05 – Mengubah Game pada Toko Game**

**F06 –**

**F07 –**

**F08 – Membeli Game**

Subprogram ini berfufungsi untuk melayani user dalam pembelian Game. Game yang telah dibeli akan masuk ke list Game yang dimiliki User. Game hanya dapat dibeli user yang sama sebanyak satu kali. Terdapat 1 parameter yang wajib diisi pada prosedur ini, yaitu ID Game yang akan dibeli user.

**F09 – Melihat Game yang Dimiliki**

Subprogram ini memberikan daftar game yang dimiliki pengguna. Tidak ada aturan khusus untuk urutan game yang ditampilkan. Tampilkan pesan khusus ketika user tidak memiliki game.

**F10 – Mencari Game yang dimiliki dari ID dan tahun rilis**

Subprogram ini dipecah lagi menjadi beberapa subprogram untuk dapat menyelesaikan fungsinya. Terdapat subprogram yang fungsinya untuk mem-filter data game sesuai dengan game\_id serta release\_year yang dimasukkan oleh pengguna. Selain itu, terdapat subprogram yang menghitung panjang karakter maksimum dari tiap kolom data game yang ingin di-output. Subprogram ini berguna untuk memperapi output data game. Subprogram satu lagi berguna untuk memfilter game yang dimiliki pengguna, sekaligus berperan sebagai subprogram utama dalam modul F10 ini.

**F11 –**

**F12 – Top Up Saldo**

Subprogram top up saldo digunakan untuk menambahkan/mengurangi data saldo pada database user.csv, yang digunakan dengan cara meminta user yang akan di topup dan jumlah saldo yang akan di topup, dimana topup hanya valid jika total akhir saldo user lebih besar dari 0. Hanya Admin yang dapat memanggil subprogram ini.

**F13 – Melihat Riwayat Pembelian**

Subprogram melihat riwayat pembelian digunakan untuk mengeluarkan riwayat pembelian dari sang pengguna, sehingga membutuhkan masukan yaitu user yang memanggilnya, dimana subprogram ini hanya bisa dipanggil oleh user karena Admin tidak bisa membeli game. Jika user tidak pernah membeli game, riwayatnya akan kosong.

**F14 – Help**

Subprogram help digunakan untuk mengeluarkan list dari fungsi-fungsi yang dapat dipanggil, tergantung dari role user, apakah user biasa atau Admin.

**F15 – Load**

Subprogram load digunakan untuk inisialiasi folder yang akan dijadikan working database yang akan dimanipulasi oleh main program. Dimana save folder yang digunakan harus valid (termasuk dalam folder database yang sudah ada)

**F16 – Save**

Subprogram save digunakan untuk menyimpan data dari working database yang dimodifikasi oleh program kedalam file csv nya, dimana jika program ditutup sebelum memanggil subprogram save, data yang telah termodifikasi tidak akan masuk kedalam database dan akan hilang.

**F17 – Exit**

Subprogram ini adalah fungsi untuk keluar dari aplikasi. Subprogram dapat menerima huruf kecil maupun besar. Pastikan masukan valid. Kalau tidak valid, bisa tanyakan kembali pertanyaannya

# **DAFTAR PEMBAGIAN KERJA ANGGOTA KELOMPOK**

Tabel 1 Daftar Pembagian Kerja Anggota Kelompok

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Modul | Implementasi | Coder | Designer | Tester |
| F02 | **Function** register | Noel Christoffel Simbolon (16521355) | Noel Christoffel Simbolon (16521355) | Kenneth Ezekiel Suprantoni  (16521040) |
| F03 | **Function** login | Noel Christoffel Simbolon (16521355) | Noel Christoffel Simbolon (16521355) | Kenneth Ezekiel Suprantoni  (16521040) |
| F04 |  | Melvin Kent Jonathan (16521247) |  | M. Bharata Sri Prana Ludira H. (16521148) |
| F05 |  | Melvin Kent Jonathan (16521247) |  | M. Bharata Sri Prana Ludira H. (16521148) |
| F06 |  | Melvin Kent Jonathan (16521247) |  | M. Bharata Sri Prana Ludira H. (16521148) |
| F07 |  | Melvin Kent Jonathan (16521247) |  | M. Bharata Sri Prana Ludira H. (16521148) |
| F08 |  | M. Bharata Sri Prana Ludira H. (16521148) |  |  |
| F09 |  | M. Bharata Sri Prana Ludira H. (16521148) |  |  |
| F10 | **Function** filter\_str  **Function** get\_max\_char\_length  **Procedure** search\_my\_game | Noel Christoffel Simbolon (16521355) | Noel Christoffel Simbolon (16521355) | Kenneth Ezekiel Suprantoni  (16521040) |
| F11 |  | Melvin Kent Jonathan (16521247) |  | M. Bharata Sri Prana Ludira H. (16521148) |
| F12 | **Function** function\_topup  **Function** topup | Kenneth Ezekiel Suprantoni  (16521040) | Kenneth Ezekiel Suprantoni  (16521040) |  |
| F13 | **Procedure** history | Kenneth Ezekiel Suprantoni  (16521040) | Kenneth Ezekiel Suprantoni  (16521040) |  |
| F14 | **Procedure** help | Kenneth Ezekiel Suprantoni  (16521040) | Kenneth Ezekiel Suprantoni  (16521040) |  |
| F15 | Load (automatically loaded) | Kenneth Ezekiel Suprantoni  (16521040) | Kenneth Ezekiel Suprantoni  (16521040) |  |
| F16 | **Procedure** saver  **Procedure** save | Kenneth Ezekiel Suprantoni  (16521040) | Kenneth Ezekiel Suprantoni  (16521040) |  |
| F17 |  | M. Bharata Sri Prana Ludira H. (16521148) |  |  |
| B01 | **Function** encrypt  **Function** decrypt | Noel Christoffel Simbolon (16521355) | Noel Christoffel Simbolon (16521355) |  |
| B02 | **Function** magicconch | (16521040) | 16521040 |  |
| B03 |  | Melvin Kent Jonathan (16521247) |  |  |

# **CHECKLIST**

Tabel 2 Checklist Pengerjaan Modul

|  |  |  |  |
| --- | --- | --- | --- |
| Modul | Desain | Implementasi | Testing |
| F02 | ✓ | ✓ | ✓ |
| F03 | ✓ | ✓ | ✓ |
| F04 | ✓ | ✓ |  |
| F05 | ✓ | ✓ |  |
| F06 | ✓ | ✓ |  |
| F07 | ✓ | ✓ |  |
| F08 | ✓ | ✓ |  |
| F09 | ✓ | ✓ |  |
| F10 | ✓ | ✓ | ✓ |
| F11 | ✓ | ✓ |  |
| F12 | ✓ | ✓ | ✓ |
| F13 | ✓ | ✓ | ✓ |
| F14 | ✓ | ✓ | ✓ |
| F15 | ✓ | ✓ | ✓ |
| F16 | ✓ | ✓ |  |
| F17 | ✓ | ✓ |  |
| B01 | ✓ | ✓ | ✓ |
| B02 | ✓ | ✓ | ✓ |
| B03 | ✓ | ✓ |  |

# **DESAIN COMMAND UNTUK SETIAP PRIMITIF**

(berisi nama command, masukan, dan keluaran)

**F02 – Register**

command: register

input: user\_data: array of array of string; name, username, password: string

output: array of array of string

**F03 – Login**

command: login

input: user\_data: array of array of string; username, password: string

output: boolean

**F08 - Membeli Game**

command: buy\_game

input: game\_data, my\_game: array of array of string; Game\_ID: string

output: string

**F09 - Melihat Game yang Dimiliki**

command: list\_game

input: game\_data: array of array of string

output: array of array of string

**F10 - Mencari Game yang dimiliki dari ID dan tahun rilis**

command: filter\_str

input: data: array of array of string; index: integer; criteria: string

output: array of array of string

command: get\_max\_char\_length

input: filtered\_game\_data: array of array of string

output: array of integer

command: search\_my\_game

input: ownership\_data, user\_data, game\_daya: array of array of string; game\_id, release\_year: string

output: - (mengoutput hasil dari filtering ke layer pengguna)

**F12 – Topup**

Command : function\_topup

Input : username : string, balance : integer, user\_data : array of array

output : array of array

Command : topup

input data : array of array

output : array of array

**F13 – History**

Command : history

Input : hist\_data : array of array

output : none

**F14 – Help**

Command : help

input user : string, save\_folder : string

output : none

**F15 – Load**

Command : -

input : folder

output : none

**F16 – Save**

Command : saver

input : folder, data

output : none

Command : save

input : data

output : none

**F17 – Exit**

command: exit

input: x: string

output: array of array, quit()

# **DESAIN KAMUS DATA**

**KAMUS GLOBAL**

running : boolean

filenames : array of string

data : array of array of array of string

admin\_callable\_commands : array of string

user\_callable\_commands : array of string

logged\_in : boolean

command : string

hist\_data : array of array

**F02**

**Function register** (user\_data : array of array of string) -> array of array of string

**KAMUS LOKAL**

name : string

username : string

char : string

password : string

char\_pass : string

id : integer

ciphered\_password : string

role : string

balance : integer

new\_user : array

**F03**

**Procedure login** (user\_data : array of array of string) -> boolean

**KAMUS LOKAL**

username, password: string

user\_valid: boolean

**F04**

**Procedure**

**KAMUS LOKAL**

Lorem Ipsum : Dolor Sit Amet

**F05**

**Procedure**

**KAMUS LOKAL**

Lorem Ipsum : Dolor Sit Amet

**F06**

**Procedure**

**KAMUS LOKAL**

Lorem Ipsum : Dolor Sit Amet

**F07**

**Procedure**

**KAMUS LOKAL**

Lorem Ipsum : Dolor Sit Amet

**F08**

**Procedure**

**KAMUS LOKAL**

ID : string

**F09**

**Procedure**

**KAMUS LOKAL**

None

**F10 - Mencari Game yang dimiliki dari ID dan tahun rilis**

**Function** **filter\_str** (data : array of array of string, index : integer, criteria : string) -> array of array of string

**KAMUS LOKAL**

temp : array of array of string

i : integer

**Function** **get\_max\_char\_length** (filtered\_game\_data : array of array of string) -> array of integer

**KAMUS LOKAL**

filtered\_game\_data\_char\_length : array

char\_length\_list : array

L : array of string

m, n : integer

max\_length\_of\_column : integer

o : array

filtered\_game\_data\_max\_char\_length : array

**Procedure** **search\_my\_game** (ownership\_data, user\_data, game\_data : array of array of string)

**KAMUS LOKAL**

game\_id, release\_year : string { user inputted filter }

user\_game\_id : array of string

i : integer { iteration variable }

game\_data\_output : array of array of string

filtered\_game\_data\_output\_by\_game\_id : array of array of string

filtered\_game\_data\_output\_by\_release\_year : array of array of string

filtered\_game\_data : array of array of string

filtered\_game\_data\_max\_char\_length : array of integer

p : integer

q : array of array of string

**F11**

**Procedure**

**KAMUS LOKAL**

Lorem Ipsum : Dolor Sit Amet

**F12**

**Function function\_topup (input** username **:** string**, input** balance **:** integer**, input** user**\_**data **:** arrayofarray**) ->** arrayofarray

**KAMUS LOKAL**

user\_valid : boolean

line\_index, current\_balance : integer

**Function topup (input** data **:** arrayofarray**) ->** arrayofarray

**KAMUS LOKAL**

username : string

balance : integer

**F13**

**Procedure history (input** hist\_data **:** arrayof array**)**

**KAMUS LOKAL**

data\_history : array of array

i, j, k, l, character\_amount : integer

**F14**

**Procedure help (input** user **:** string**, input** save**\_**folder **:** string**)**

**KAMUS LOKAL**

is\_user\_admin : boolean

**F15**

**Procedure Load**

**KAMUS LOKAL**

parser : function

save\_folder : string

all\_folder : array

**F16**

**Procedure saver (input** folder **:** string**, input** data **:** array**)**

**KAMUS LOKAL**

path : string

exist : boolean

**F17**

**Procedure**

**KAMUS LOKAL**

x : string

**B01**

**Function encrypt** (password : string) -> string

**KAMUS LOKAL**

a, b : integer

ciphered : string

**Function decrypt** (ciphered : string) -> string

**KAMUS LOKAL**

a, b : integer

i : integer

password : string

**B02**

**Function magicconch()**

**KAMUS LOKAL**

x, a, c, m, state : integer

**B03**

**Procedure**

**KAMUS LOKAL**

Lorem Ipsum : Dolor Sit Amet

**FUNGSI TAMBAHAN**

**Procedure**

**KAMUS LOKAL**

Lorem Ipsum : Dolor Sit Amet

# **DESAIN DEKOMPOSISI ALGORITMIK DAN FUNGSIONAL PROGRAM**

**F02 - Register**

A picture containing shape

Description automatically generated

Gambar 1 Flowchart modul 02

**F03 – Login**

Diagram

Description automatically generated

Gambar 2 Flowchart modul 03

**F04 – Menambah Game ke Toko Game**

Gambar 6.4 Flowchart untuk modul 4

**F05 – Mengubah Game pada Toko Game**

Gambar 6.5 Flowchart untuk modul 5

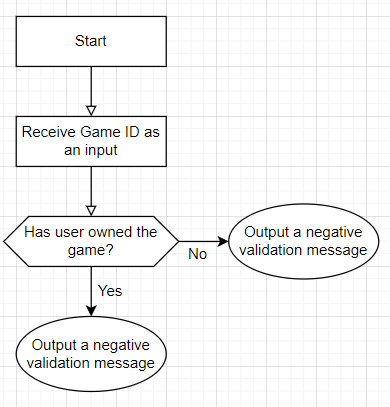
**F06 – Mengubah Stok Game di Toko**

Gambar 6.6 Flowchart untuk modul 6

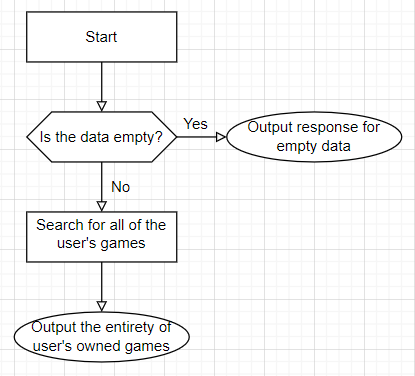
**F07 – Listing Game di Toko Berdasarkan ID, Tahun Rilis dan Harga**

Gambar 6.7 Flowchart untuk modul 7

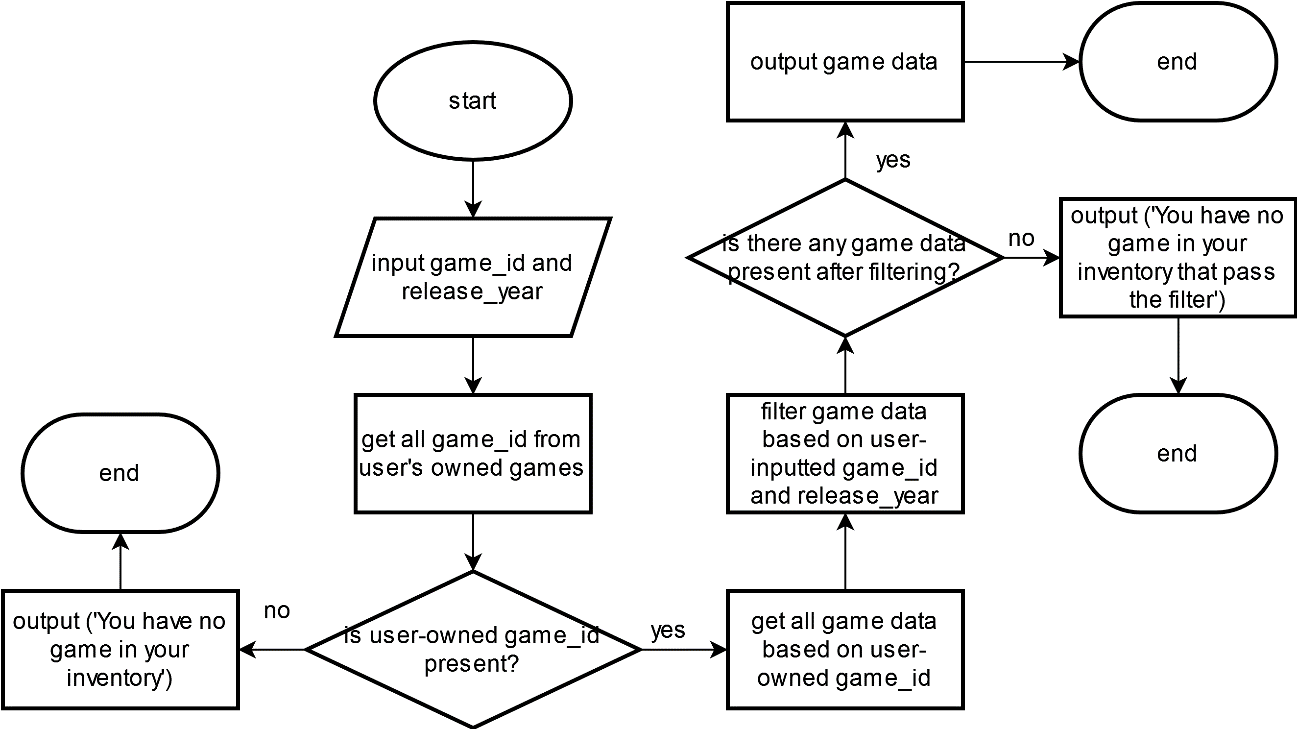
**F08 – Buy Game**



**F09 – List Game**



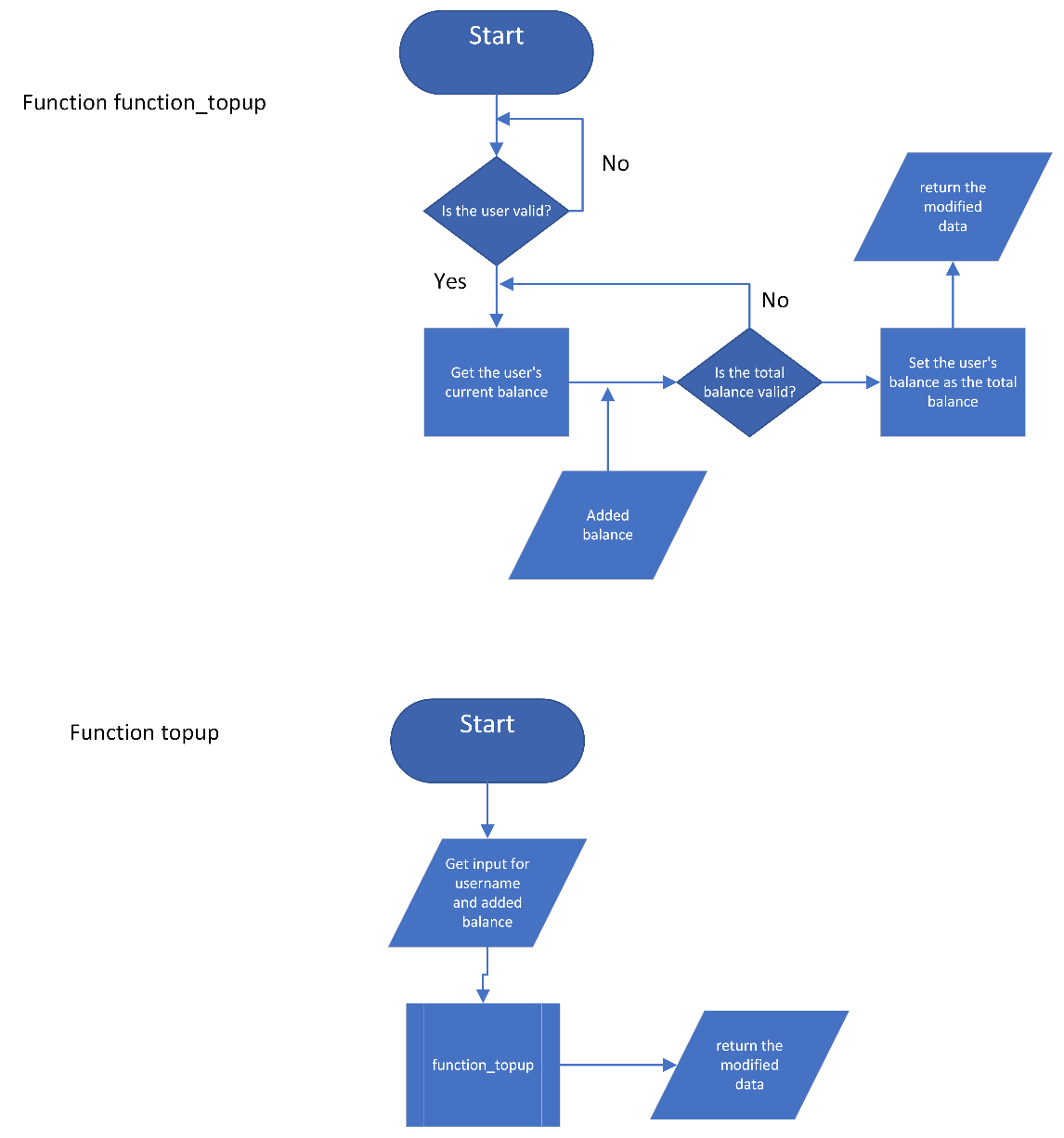
**F10 – Mencari Game yang dimiliki dari ID dan tahun rilis**



**F11 – Mencari Game di Toko dari ID, Nama Game, Harga, Kategori dan Tahun Rilis**

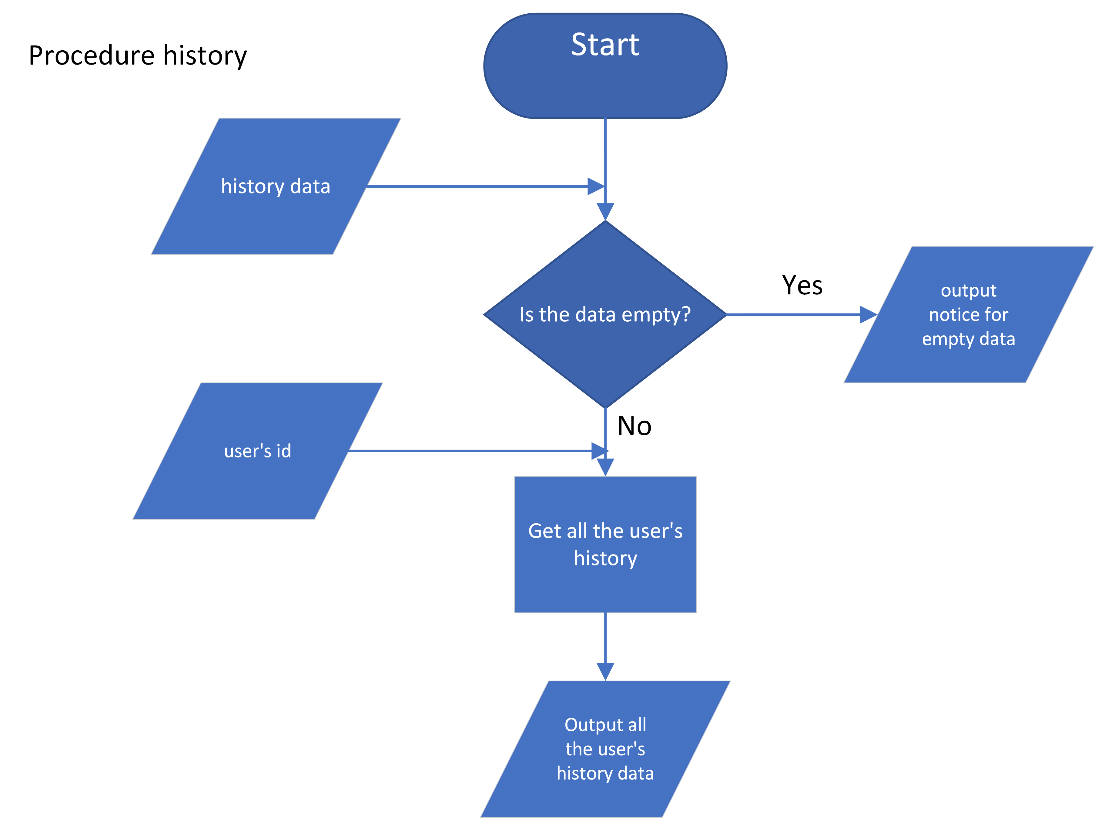
Gambar 6.11 Flowchart untuk modul 11

**F12 – Topup**

****

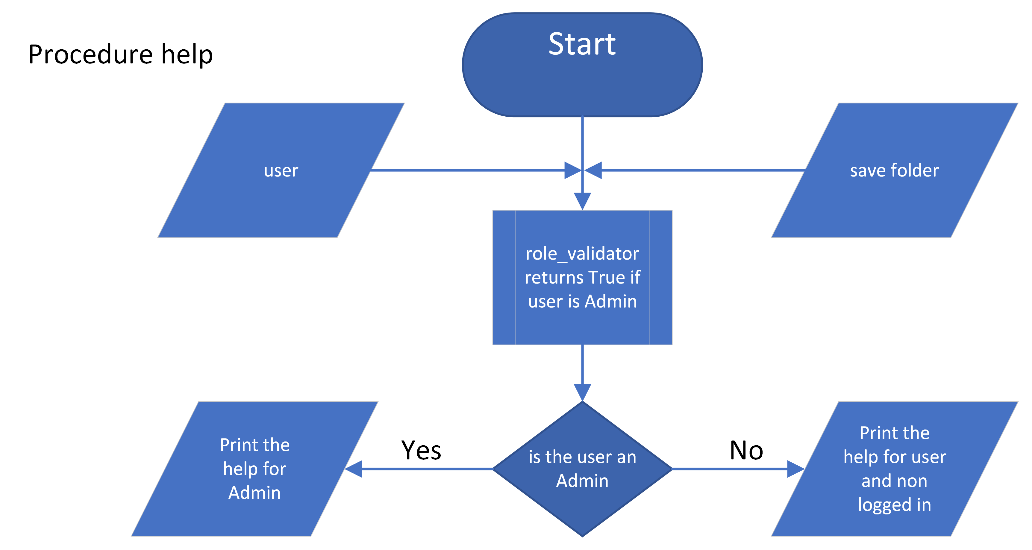
Gambar 6.12 Flowchart untuk modul 12

**F13 – History**

****

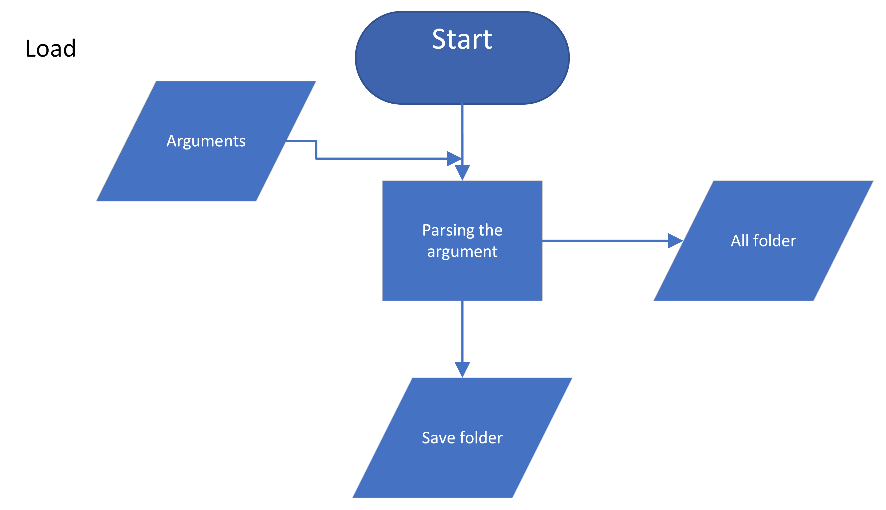
Gambar 6.13 Flowchart untuk modul 13

**F14 – Help**

****

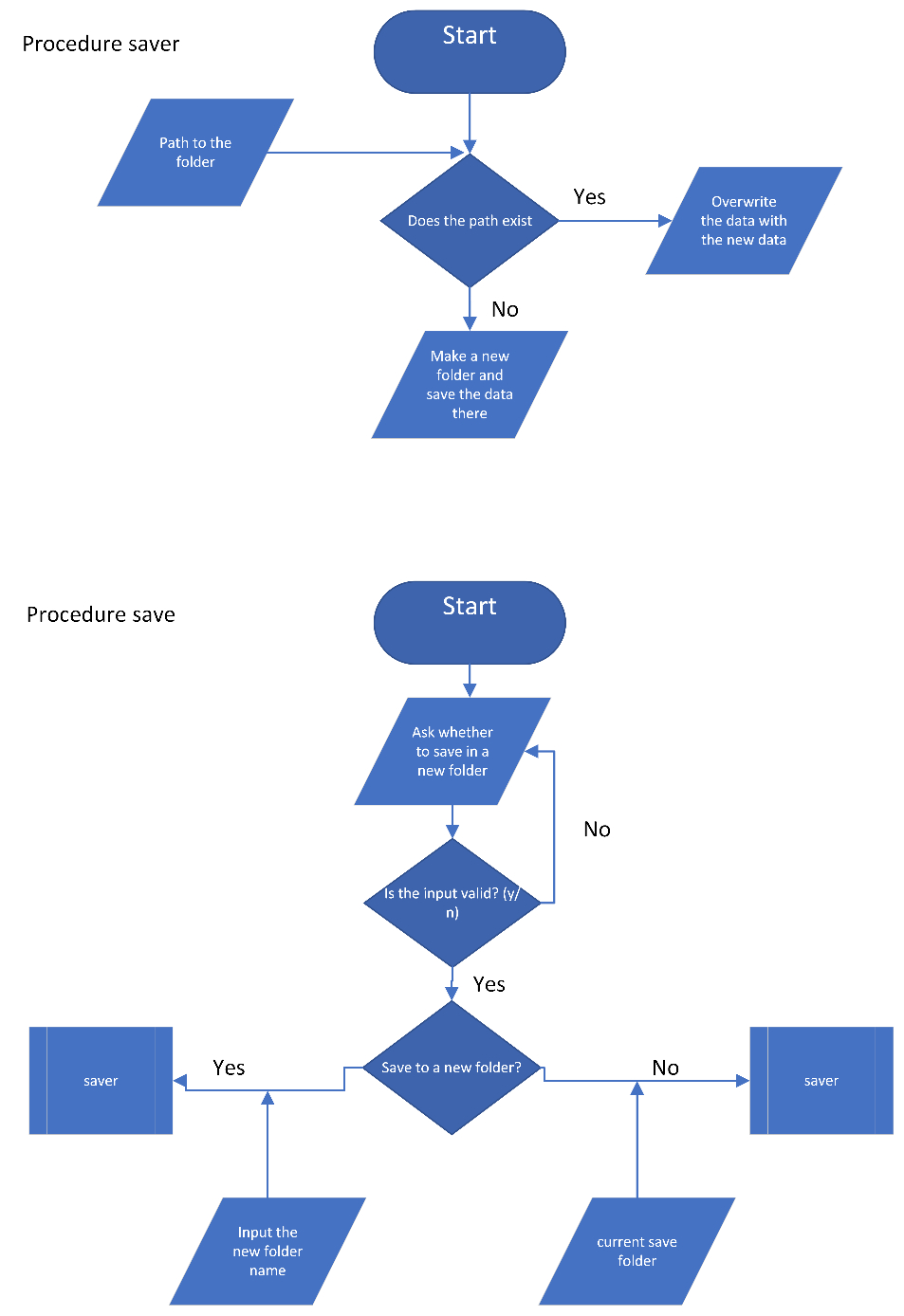
Gambar 6.14 Flowchart untuk modul 14

**F15 – Load**

****

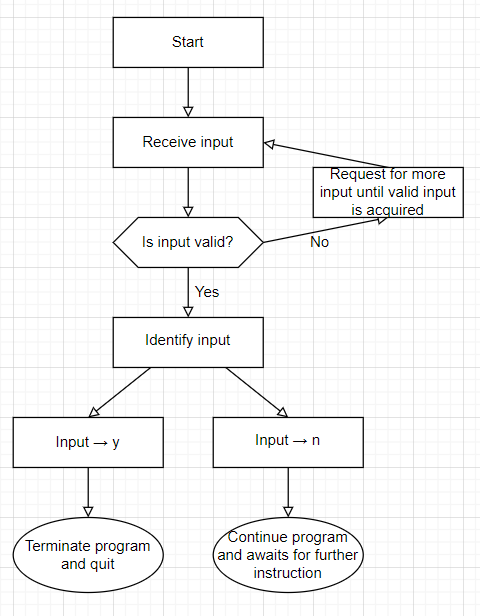
Gambar 6.15 Flowchart untuk modul 15

**F16 – Save**

****

Gambar 6.16 Flowchart untuk modul 16

**F17 – Exit**



# **SPESIFIKASI UNTUK SETIAP MODUL YANG DIBUAT MAIN PROGRAM**

**F02 – Register**

|  |
| --- |
| **DICTIONARY**  function length (input) -> integer  { Function to calculate the length of an object. }  function append (list\_ : array, input) -> array  { Function to append an input to a list. }  function encrypt (password : string) -> string  { Encrypts user password using the Affine cipher. }  **function** register (user\_data : array of array of string) -> array of array of string  { Function to add a list of id, username, name, ciphered password, role, and balance of user  to the loaded user.csv data on the main program (GUI.py). }  **LOCAL DICTIONARY**  name, username, ciphered\_password, role, char, password, char\_pass : string  id, balance : integer  **ALGORITHM**  input (name)  input (username)  { Loops until the username is valid }  while (True) do  try  { Username valid characters (-, \_, 0-9, A-Z, a-z) validation }  char traversal username  if (not (ord(char) = 45 or ord(char) = 95 or 48 <= ord(char) <= 57 or  65 <= ord(char) <= 90 or 97 <= ord(char) <= 122)) then { -, \_, 0-9, A-Z, a-z respectively }  output ('Username is not valid. Please only use letters, numbers, underscore (\_), and dash (-).')  raise ValueError  { Checks if the username is already present }  { Loop for every entry in user.csv excluding the first line }  i traversal [1..length(user\_data) - 1]  if (username =s user\_data[i][1]) then  output (f'Username "{username}" already exists, please select a different username.\n')  raise ValueError  except ValueError  input (username)  else  output ('Username is available!')  break  input (password)  { Loops until the password is valid }  while (True) do  try  { Password validation to not break user.csv }  char\_pass traversal password  if (char\_pass = ';') then  output ('Password must not contain semicolon (;)')  raise ValueError  except ValueError  input (password)  else  output ('Password is valid!')  break  id <- length(user\_data)  ciphered\_password <- encrypt(password)  role <- 'User' { Register can only add a user, not admin }  balance <- 0 { Initial balance is always 0 }  new\_user <- [id, username, name, ciphered\_password, role, balance]  user\_data <- append(user\_data, new\_user)  -> user\_data |

**F03 – Login**

|  |
| --- |
| **DICTIONARY**  function decrypt (ciphered : string) -> string  { Decrypts ciphered user password using the Affine cipher. }  function length (input) -> integer  { Function to calculate the length of an object. }  user\_line\_index : integer { global variable on what index is the current user's username stored in user.csv }  **function** login (user\_data : array of array of string) -> boolean  { Returns True if the username and password is correct and it is on the database.  Returns False otherwise. }  **LOCAL DICTIONARY**  user\_valid : boolean { username present in database or not }  username, password : string  **ALGORITHM**  input (username)  input (password)  user\_valid <- False  { Checks if the username is present in database }  { Loop for every line in file user.csv (index 3 on folder save) (ignore the first line) }  i traversal [1..length(user\_data) - 1]  if (username = user\_data[i][1]) then  user\_valid <- True  global user\_line\_index  user\_line\_index <- i  if (user\_valid) then  if (decrypt(user\_data[user\_line\_index][3]) = password) then  output (f'Welcome to BNMO, {username}!')  -> True  else { not (decrypt(user\_data[user\_line\_index][3]) = password) }  output ('Username not found or wrong password')  -> False  else { not (user\_valid) }  output ('Username not found or wrong password')  -> False |

**F04 – Menambah Game ke Toko Game**

|  |
| --- |
| **DICTIONARY**  **function** length (input) -> integer  { Function to calculate the length of an object. }    **function** append (list\_ : array, input) -> array  { Function to append an input to a list. }      {not to be imported}  **function** index\_constructor (game\_data : array of array of string) -> array of array of string  {I.S. game\_data array of is ordered based on game ID}  {F.S. new index is generated and returned}    **LOCAL DICTIONARY**  prevoius\_number, new\_number : integer  new\_index : string    **ALGORITHM**  {Fetching the integers of the last Game ID}  previous\_number <- int(game\_data[length(game\_data)-1][0][4]) \* 100 + int(game\_data[length(game\_data)-1][0][5]) \* 10 + int(game\_data[length(game\_data)-1][0][6])    new\_number <- previous\_number + 1  if (new\_number < 10) then  new\_index <- "GAME00" + str(new\_number)  else (if new\_number < 100) then  new\_index <- "GAME0" + str(new\_number)  else  new\_index <- "GAME" + str(new\_number)    -> new\_index    {not to be imported}  **function** new\_game (game\_data : array of array of string) -> array of array of string  **LOCAL DICTIONARY**  complete : boolean  name, category, release\_year : string  price, stock : integer  new\_index : string  new\_data : array of string    **ALGORITHM**  {I.S. game\_data is defined and have all the atrributes}  {F.S. new game data is collected and validated, index is generated automatically, new data is returned}    complete <- False  while (complete = False) do # loop for input completeness validation  input(name, category, release\_year, price, stock)    {Input validation}  if (length(name) = 0) or (length(category) = 0) or (length(release\_year) = 0) or (length(price) = 0) or (length(stock) = 0) then  output("Please insert all of the game information to be saved by BNMO.")  else  complete <- True    new\_index <- index\_constructor(game\_data) # generating id for the new game by fetching the latest id from the database + 1  new\_data <- [new\_index , name, category, release\_year, int(price), int(stock)]    -> new\_data      **function** add\_game (game\_data : array of array of string) -> array of array of string  {I.S game\_data is defined and have all the atrributes}  {F.S game\_data array is returned with validated new game data}    **LOCAL DICTIONARY**  new\_data : array of string  game\_data : array of array of string    **ALGORITHM**  new\_data <- new\_game(game\_data)  game\_data <- append(game\_data, new\_data)  output("Congratulations! Adding game succeded", new\_data[1] + "." )    -> game\_data |

**F05 – Mengubah Game pada Toko Game**

**F06 – Mengubah Stok Game di Toko**

**F07 – Listing Game di Toko Berdasarkan ID, Tahun Rilis dan Harga**

**F08 – Membeli Game**

**DICTIONARY**

function is (list : list) -> (list : list) -> string -> string

**LOCAL DICTIONARY**

game\_data: list of lists

my\_game: list of lists

ID : string

**ALGORITHM**

ID <- String

if ID in game\_data then

if ID in my\_game then

output(“You have owned that game!”)

else

output(“Game is succesfully bought!”)

else

output(“Game doesn’t exist”)

**F09 – Melihat Game yang Dimiliki**

**DICTIONARY**

function is (list : list) -> (list : list)

**LOCAL DICTIONARY**

Game\_data : list of lists

**ALGORITHM**

If game\_data length = 0

output(“You haven’t bought any game”)

else:

output(game\_data)

**F10 – Mencari Game yang dimiliki dari ID dan tahun rilis**

|  |
| --- |
| **DICTIONARY**  function length (input) -> integer  { Function to calculate the length of an object. }  function append (list\_ : array, input) -> array  { Function to append an input to a list. }  function enum (list\_ : array, start=0)  { Works the same way as the built-in function enumerate(). Returns an enumerate object. }  user\_line\_index : integer { global variable on what index is the current user's username stored in user.csv }  { not to be imported }  [ data is filtered list, index is the location of the targetted attribute, criteria is the previously asked input by search\_game\_at\_store procedure }  **function** filter\_str (data : array of array of string, index : integer, criteria : string) -> array of array of string  { Function to create a list filtered by a string attribute }  **LOCAL DICTIONARY**  temp : array of array of string  i : integer { iteration variable }  **ALGORITHM**  if (criteria = "") then  temp <- data  else { not (criteria = "") }  temp <- [] { temp is for hosting matching datas }  { traversing to find matching attribute and appending the list to temp }  i traversal [0..length(data) - 1]  if (criteria = data[i][index]) then  temp <- append(temp, data[i])  -> temp  { not to be imported }  **function** get\_max\_char\_length (filtered\_game\_data : array of array of string) -> array of integer  { Function to get the maximum character length for each column in the filtered game data  (i.e. filtered game.csv according to kepemilikan.csv and user-inputted game\_id and release\_year) }  **LOCAL DICTIONARY**  filtered\_game\_data\_char\_length : array  char\_length\_list : array  L : array of string  m, n : integer  max\_length\_of\_column : integer  o : array  filtered\_game\_data\_max\_char\_length : array  **ALGORITHM**  filtered\_game\_data\_char\_length <- []  L traversal filtered\_game\_data  char\_length\_list <- []  m traversal [0..4] { Don't index the stock of game }  { char\_length\_list is a list of character length for each game entry }  char\_length\_list <- append(char\_length\_list, length(L[m]))  { filtered\_game\_data\_char\_length is the complete list of list of character length for the filtered game data }  filtered\_game\_data\_char\_length <- append(filtered\_game\_data\_char\_length, char\_length\_list)  filtered\_game\_data\_max\_char\_length <- []  n traversal [0..4]  max\_length\_of\_column <- 0  o traversal filtered\_game\_data\_char\_length  if (o[n] > max\_length\_of\_colum) then  max\_length\_of\_column <- o[n]    filtered\_game\_data\_max\_char\_length <- append(filtered\_game\_data\_max\_char\_length, max\_length\_of\_column)  -> filtered\_game\_data\_max\_char\_length  { ownership\_data is the kepemilikan.csv of a save folder, user\_data is the user.csv, while game\_data is the game.csv }  **procedure** search\_my\_game (ownership\_data, user\_data, game\_data : array of array of string)  { Procedure that prints user-owned games based on its ID and release year. }  **LOCAL DICTIONARY**  game\_id, release\_year : string { user inputted filter }  user\_game\_id : array of string  i : integer { iteration variable }  game\_data\_output : array of array of string  filtered\_game\_data\_output\_by\_game\_id : array of array of string  filtered\_game\_data\_output\_by\_release\_year : array of array of string  filtered\_game\_data : array of array of string  filtered\_game\_data\_max\_char\_length : array of integer  p : integer  q : array of array of string  **ALGORITHM**  input (game\_id.upper())  input(release\_year)  user\_game\_id <- [] { All Game ID of the currently logged in user }  { Loop for every entry in kepemilikan.csv excluding the first line }  i traversal [1..length(ownership\_data) - 1]  { if user id in kepemilikan.csv == user id of currently logged in user, then append the game id of the game with matchin user id }  if (ownership\_data[i][1] = user\_data[user\_line\_index][0]) then  user\_game\_id <- append(user\_game\_id, ownership\_data[i][0])  if (length(user\_game\_id) = 0) then  output ('You have no game in your inventory')    else { not (length(user\_game\_id) = 0) }    game\_data\_output <- []  j traversal user\_game\_id  { Loop for every entry in game.csv excluding the first line }  k traversal [1..length(game\_data) - 1]  if (j = game\_data[k][0]) then { if user game id matches game id in game.csv }  game\_data\_output <- append(game\_data\_output, game\_data[k])  { Filter based on game\_id and release\_year }  filtered\_game\_data\_output\_by\_game\_id <- filter\_str(game\_data\_output, 0, game\_id)  filtered\_game\_data\_output\_by\_release\_year <- filter\_str(filtered\_game\_data\_output\_by\_game\_id, 3, release\_year)  filtered\_game\_data <- filtered\_game\_data\_output\_by\_release\_year  if (length(filtered\_game\_data) = 0) then  output ('You have no game in your inventory that pass the filter')    else { not (length(filtered\_game\_data) = 0) }    filtered\_game\_data\_max\_char\_length <- get\_max\_char\_length(filtered\_game\_data)  output ('\nGames in your inventory that meet the filter:')  p, q traversal enum(filtered\_game\_data, start=1)  output (f'{p}. {q[0].ljust(filtered\_game\_data\_max\_char\_length[0])} | {q[1].ljust(filtered\_game\_data\_max\_char\_length[1])} | {q[4].ljust(filtered\_game\_data\_max\_char\_length[4])} | {q[2].ljust(filtered\_game\_data\_max\_char\_length[2])} | {q[3].ljust(filtered\_game\_data\_max\_char\_length[3])}') |

**F12 – Topup**

|  |
| --- |
| **DICTIONARY**  {  function length (a) -> integer  }  **FUNCTION/PROCEDURE DEFINITION**  **Function** function\_topup (input username : string, input balance : integer, input user\_data : array of array) -> array of array  {Function to topup the user's balance}    **LOCAL DICTIONARY**  {  user\_valid : boolean  line\_index, current\_balance : integer  }  **ALGORITHM**  user\_valid <- False  line\_index <- 0  i traversal [2..length(user\_data)]  {Checks if the user is a valid user or not}  if username = user\_data[i][2] then  user\_valid <- True  line\_index <- i    if user\_valid = True then  current\_balance <- user\_data[line\_index][6]  if balance + current\_balance < 0 then  output("Input not valid")  else  current\_balance <- current\_balance + balance  user\_data[line\_index][6] <- current\_balance  -> user\_data    else  output("Username", username, "not found")  **Function** topup (input data : array of array) -> array of array  {Function to get input and inputs it into the function\_topup}  **LOCAL DICTIONARY**  {  username : string  balance : integer  }  **ALGORITHM**  input(username, balance)  data <- function\_topup(username, balance, data)  -> data |

**F13 – History**

|  |
| --- |
| **DICTIONARY**  {  history\_data : array of array    function length (a) -> integer  function append (a, array) -> array  }  **FUNCTION/PROCEDURE DEFINITION**  **Procedure** history (input hist\_data : array of array)  {Procedure to print the content of riwayat.csv array in the working data\_history (temporary data matrix)}  {I.S. hist\_data is defined and not empty (minimum 1 element)  F.S. hist\_data is printed}  **LOCAL DICTIONARY**  {  user\_hist\_data : array of array  data\_history : array of array  i, j, k, l, character\_amount : integer  }  **ALGORITHM**    {Loop to check for all the user's history data}  i traversal [1..length(history\_data)]  if history\_data[i][4] = user\_id then  user\_hist\_data += history\_data[i]  if length(hist\_data) = 1 then  output("Sorry, you haven't bougth any game yet. Enter buy\_game to buy some game.)  else  {Generating a temporary list to host data without changing the original source}  data\_history <- array [1..length(hist\_data)] of "\*"  i traversal [2..length(hist\_data)]  data\_history[i] <- hist\_data[i+1]    data\_history <- append (["HEADING"], data\_history)  {Generating parsing for non-empty data\_history list}  i traversal [1..length(data\_history)]  output(i, ".", end: " ")    j traversal [1..6]  output(data\_history[i][j], end: "")  character\_amount <- 0  k traversal [1..length(data\_history)]  if length(data\_history[k][j]) > character\_amount then  character\_amount <- length(data\_history[k][j])    l traversal [1..character\_amount-length(data\_history[i][j])]  output(" ", end: "")  output("| ", end:"")  output("\n") |

**F14 – Help**

|  |
| --- |
| **DICTIONARY**  {  function is\_admin (user, save\_folder) -> Boolean  }  **FUNCTION/PROCEDURE DEFINITION**  **Procedure** help (input user : string, input save\_folder : string)    {Procedure to print the instructions for the main program}    {I.S. user is defined, save-folder is defined, role\_validator function is defined  F.S. Help instructions are printed}  **LOCAL DICTIONARY**  {  is\_user\_admin : boolean  }  **ALGORITHM**  is\_user\_admin = is\_admin (user, save\_folder)  if is\_user\_admin = True then  output("========== HELP ==========")  output("")  output("1. register - Register a new user")  output("2. login - Log in to the program")  output("3. add\_game - Adding a game to the database")  output("4. change\_game - Changing a game in the database")  output("5. change\_stock - Changing the stock of a game in the database")  output("6. list\_available\_game - Gives a list of all the available game in the store")  output("7. search\_at\_store - Searches the store for a game")  output("8. topup - Top ups the balance of a user")  output("9. help - Prints this menu")  output("10. save - Saves the current working database")  output("11. exit - Exits the program")  print("12. magicconch : Hears what the great magic conch has to say")  print("13. tictactoe : Play TicTacToe")  else  output("========== HELP ==========")  output("")  output("before logging in:")  output("1. login - Log in to the program")  output("2. help - prints this menu")  output("")  output("after logging in:")  output("1. list\_available\_game - Gives a list of all the available game in the store")  output("2. buy\_game - Buys a game with the current balance")  output("3. list\_my\_game - Lists owned games")  output("4. search\_my\_game - Searches owned games")  output("5. search\_at\_store - Searches the store for a game")  output("6. history - Prints the transaction history")  output("7. help - Prints this menu")  output("8. save - Saves the current working database")  output("9. exit - Exits the program")  print("10. magicconch : Hears what the great magic conch has to say")  print("11. tictactoe : Play TicTacToe") |

**F15 – Load**

|  |
| --- |
| **DICTIONARY**  {    function ArgumentParser  function walk  }  **ALGORITHM**  parser <- ArgumentParser()  parser.add\_argument("folder", help="the save file that is want to be loaded")  args <- parser.parse\_args()  save\_folder <- args.folder  all\_folder <- next(walk("Database"))[1] |

**F16 – Save**

|  |
| --- |
| **DICTIONARY**  {  procedure writeline  function os  save\_folder : string  }  **FUNCTION/PROCEDURE DEFINITION**  **Procedure** saver (input folder : string, input data : array)    {Procedure to save the data in the program to the database}    {I.S. folder is defined, data (matrix) is defined  F.S. The working database is saved to the csv}  **LOCAL DICTIONARY**  {  path : string  exist : boolean  }  **ALGORITHM**  path <- "Database/{folder}"  exist <- os.path.exists(path)  if exist then  {overwrite the data}  writeline(folder, "game.csv", data[0])  writeline(folder, "kepemilikan.csv", data[1])  writeline(folder, "riwayat.csv", data[2])  writeline(folder, "user.csv", data[3])  else  {make a new folder}  open(path/files, "w") # Make a new file for every files  writeline(folder, "game.csv", data[0])  writeline(folder, "kepemilikan.csv", data[1])  writeline(folder, "riwayat.csv", data[2])  writeline(folder, "user.csv", data[3])  **Procedure** save (input data : list)  {Procedure to ask whether to save in the same save folder or a different one}    {I.S. saver is defined, data is defined  F.S. the saver runs with a folder defined (new/existing)}  **LOCAL DICTIONARY**  {  is\_new\_folder, new\_folder, folder : string  }  **ALGORITHM**  is\_new\_folder <- input("Do you wish to save to a new folder? (y/n) ")  while (is\_new\_folder != "y") and (is\_new\_folder != "Y") and (is\_new\_folder != "n") and (is\_new\_folder != "N") do  {Input Validation}  output("Unknown input. Please choose between (y/n)")  is\_new\_folder <- input("Do you wish to save to a new folder? (y/n) ")    if (is\_new\_folder == "y") or (is\_new\_folder == "Y") then  new\_folder <- input("folder name: ")  saver(new\_folder, data)  elif (is\_new\_folder == "n") or (is\_new\_folder == "N") then  folder <- save\_folder  saver(folder, data) |

**F17 – Exit**

**DICTIONARY**

function is string -> quit()

**LOCAL DICTIONARY**

x : string

**ALGORITHM**

X <- String

if (x = Y) or (x = y) then

quit()

if (x = N) or (x = n) then

Ø

-> quit()

**B01 – Cipher**

|  |
| --- |
| **DICTIONARY**  function is\_lower (string : string) -> boolean  { Function to check if a string consists entirely of lowercase letters. }  **function** encrypt (password : string) -> string  { Encrypts user password using the Affine cipher. }  **LOCAL DICTIONARY**  a, b : integer { cipher keys }  ciphered : string { ciphered password }  **ALGORITHM**  { Hardcoded key }  a <- 17  b <- 9  ciphered <- ''    { e(x) = (ax + b) mod m }  char traversal password  if (97 <= ord(char) <= 122 or 65 <= ord(char) <= 90) then  if (is\_lower(char)) then  ciphered <- ciphered + chr(((a \* (ord(char) - 97) + b) % 26) + 97)  else { is\_upper(char) }  ciphered <- ciphered + chr(((a \* (ord(char) - 65) + b) % 26) + 65)  else { not (97 <= ord(char) <= 122 or 65 <= ord(char) <= 90) }  ciphered <- ciphered + char  -> ciphered  **function** decrypt (ciphered : string) -> string  { Decrypts ciphered user password using the Affine cipher. }  **LOCAL DICTIONARY**  a, b : integer { cipher keys }  i : integer  password : string  **ALGORITHM**  { Hardcoded key }  a <- 17  b <- 9  i <- 0  password <- ''  { Finding a^(-1) which is the multiplicative inverse of a }  multiplicative\_inverse <- None  while (multiplicative\_inverse = None) do  if (((i \* 26) + 1) / a = ((i \* 26) + 1) // a) then  multiplicative\_inverse <- int(((i \* 26) + 1) / a)  break  else { not (((i \* 26) + 1) / a = ((i \* 26) + 1) // a) }  i <- i + 1    { d(x) = a^(-1)(x - b) mod m }  char traversal ciphered  if (97 <= ord(char) <= 122 or 65 <= ord(char) <= 90) then  if (is\_lower(char)) then  password <- password + chr(((multiplicative\_inverse \* ((ord(char) - 97) - b)) % 26) + 97)  else { is\_upper(char) }  password <- password + chr(((multiplicative\_inverse \* ((ord(char) - 65) - b)) % 26) + 65)  else { not (97 <= ord(char) <= 122 or 65 <= ord(char) <= 90) }  password <- password + char  -> password |

**B02 – Magic conch**

|  |
| --- |
| **DICTIONARY**  {  function time () -> integer  }  **FUNCTION/PROCEDURE DEFINITION**  **Function** magicconch () -> string  {Function that generates a random number with LCG and returns a string based on the random number}    **LOCAL DICTIONARY**  {  x, a, c, m, state : integer  }  **ALGORITHM**  x <- time.time()  a <- 3  c <- 1  m <- 7  state <- round(((a\*x) + c) mod 7)  {States and outputs}  if (state == 0) then  -> "Coba lagi."  else if (state == 1) then  -> "Ya."  else if (state == 2) then  -> "Tidak."  else if (state == 3) then  -> "Mungkin."  else if (state == 4) then  -> "Jangan deh."  else if (state == 5) then  -> "Tanya lagi nanti."  else if (state == 6) then  -> "Terserah dah."  else if (state == 7) then  -> "Coba tanya doswal."  else  do nothing |

**B03 – TicTacToe**

|  |
| --- |
| **DICTIONARY**  { not to be imported }  **procedure** ask\_location(input/output matrix : array of array of strings, input pawn : string)  **LOCAL ALGORITHM**  valid : boolean  x, y : integer    **ALGORITHM**  { Procedure to ask input for pawn location and validate it. }  valid <- False  while (valid = False) do  output("["+ pawn + "] turn: ")  input (x, y)  # Location validation  if not ((1<=x<=3) and (1<=y<=3) ) then {the location does not exist}  output("Invalid location. Please try again!")  else {the location exists}  if (matrix[y-1][x-1] != "#") then {the location is alrady occupied}  output("Location is already filled. Please try again!")  else {teh location is empty}  matrix[y-1][x-1] <- pawn  valid <- True  { not to be imported }  function win\_checker(matrix : array of array of strings, pawn : string) -> string  {Function to return state of winning of a pawn}  **LOCAL DICTIONARY**  win : string  **ALGORITHM**  win <- "" {win = "" --> pawn haven't won yet}  {horizontal win checker}  if ((matrix [0][0] = pawn) and (matrix [0][1] = pawn) and (matrix [0][2] = pawn)) or ((matrix [1][0] = pawn) and (matrix [1][1] = pawn) and (matrix [1][2] = pawn)) or ((matrix [2][0] = pawn) and (matrix [2][1] = pawn) and (matrix [2][2] = pawn)) then  win <- "horizontally"    {vertical win checker}  else if ((matrix [0][0] = pawn) and (matrix [1][0] = pawn) and (matrix [2][0] = pawn)) or ((matrix [0][1] = pawn) and (matrix [1][1] = pawn) and (matrix [2][1] = pawn)) or ((matrix [0][2] = pawn) and (matrix [1][2] = pawn) and (matrix [2][2] = pawn)) then  win <- "vertically"    # diagonal win checker  else if ((matrix [0][0] = pawn) and (matrix [1][1] = pawn) and (matrix [2][2] = pawn)) or ((matrix [0][2] = pawn) and (matrix [1][1] = pawn) and (matrix [2][0] = pawn)) then  win <- "diagonally"  -> win  { not to be imported }  procedure status(input/output matrix : array of array of strings)  {Procedure to print out board status.}  **LOCAL DICTIONARY**  i,j : integer  **ALGORITHM**  output("=============")  output("Board Status:")  {Generate parsing for matrix}  i traversal [0..2]  output("|", end=" ")  j taversal [0..2]  output(matrix[i][j], end=" ")  output("|", end=" ")  output("")  **procedure** tictactoe ()  {Procedure to simulate tic tac toe game}  **LOCAL DICTIONARY**  matrix : array of array of characters  turn : integer  pawn, string : string  **ALGORITHM**  matrix <- [["#", "#", "#"], ["#", "#", "#"], ["#", "#", "#"]]  turn <- 0  while (turn<=9) do  turn <- turn + 1  if turn%2 = 1 then  pawn <- "X"  else  pawn <- "O"    status(matrix) { print out board status }  ask\_location(matrix, pawn) { ask for user input of pawn location }  { check if pawn wins }  win <- win\_checker(matrix, pawn)  if win != "" then {(win = "")--> meaning pawn haven't won yet }  status(matrix)  if (win = "horizontally") then  output(pawn, "won horizontally. Victory applies to other row.")  else if (win = "vertically") then  output(pawn, "won vertically. Victory applies to other column.")  else {win = "diagonally"}  output(pawn, "won diagonally. Victory applies to the opposite diagonal.")  break  else { win == "" --> pawn haven't won }  pass  { turn == 9 is the last turn; tie statement will be skipped if there is already a winner }  if (turn = 9) then  status(matrix)  output("Tie. There is no winner.")  break |

# **HASIL SCREENSHOT PENGUJIAN PROGRAM BERDASARKAN FITUR-FITUR PADA SPESIFIKASI**

**F02 – Register**

Input

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

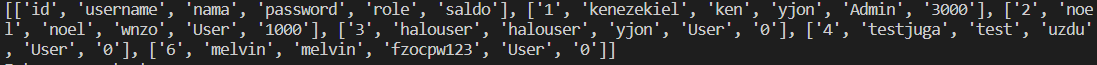
Gambar 9.2.1 Cara memanggil prosedur dalam modul F02

Sebuah gambar berisi teks

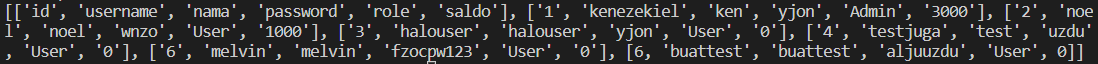
Deskripsi dibuat secara otomatis

Gambar 9.2.2 Input untuk modul F02

Output



Gambar 9.2.3 Sebelum modifikasi register



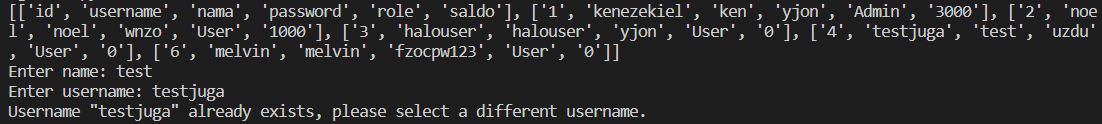
Gambar 9.2.4 Output untuk modul F02

Validasi

Sebuah gambar berisi teks

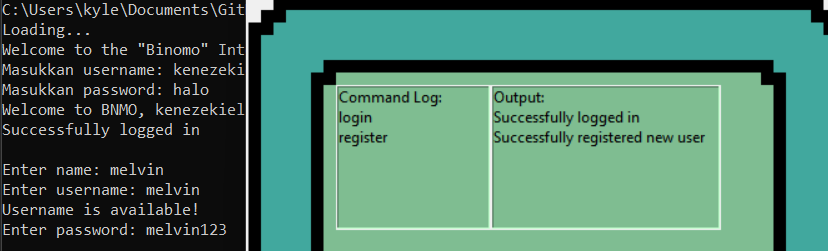
Deskripsi dibuat secara otomatis

Gambar 9.2.5 Testing validasi input untuk input modul F02



Gambar 9.2.6 Testing validasi username untuk input modul F02

Implementasi di Main Program



Gambar 9.2.7 Contoh implementasi modul F02 pada main program

**F03 – Login**

Input

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

Gambar 9.3.1 Cara memanggil fungsi dalam modul F03



Gambar 9.3.2 Input untuk modul F03

Output

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

Gambar 9.3.3 Output untuk modul F03

Validasi

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

Gambar 9.3.4 Testing validasi untuk input modul F03

Implementasi di Main Program

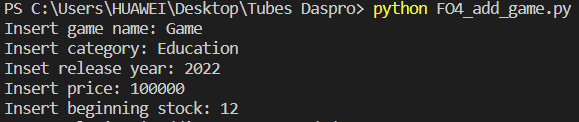
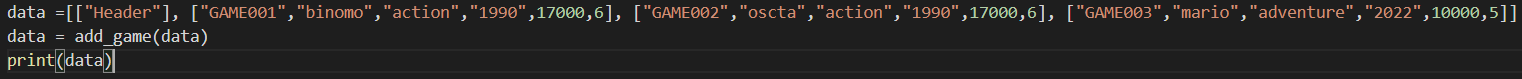
Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

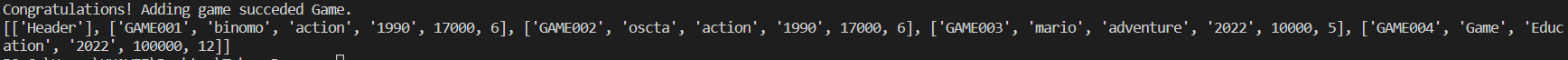
Gambar 9.3.5 Contoh implementasi modul F03 pada main program

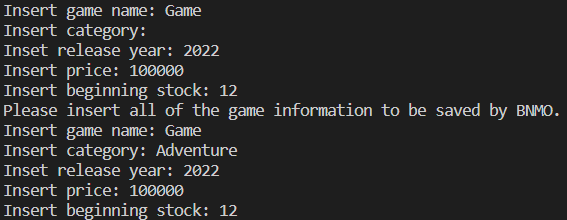
**F04 – Menambahkan Game**

Input

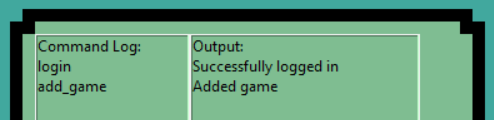
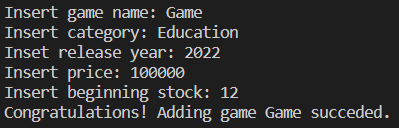


Output

Validasi

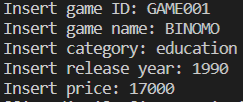
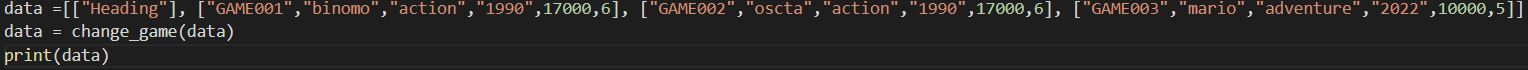


Implementasi di Main Program



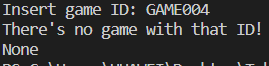
**F05 – Mengubah Game**

Input

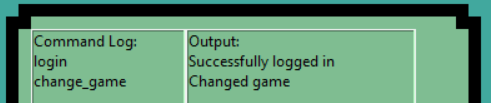
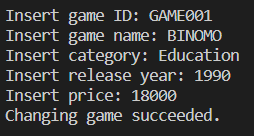


Output

Validasi

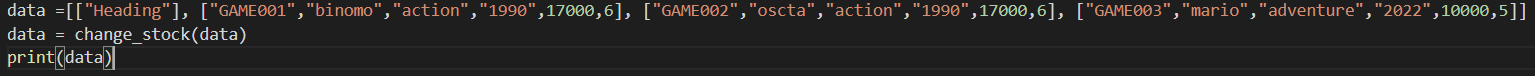


Implementasi di Main Program



**F06 – Mengubah Stok Game**

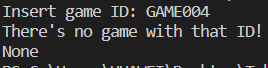
Input



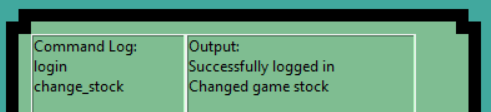
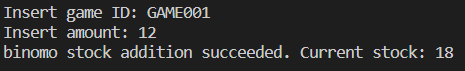
Output



Validasi



Implementasi di Main Program

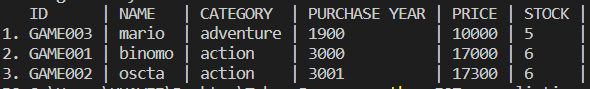


**F07 – List Available Game**

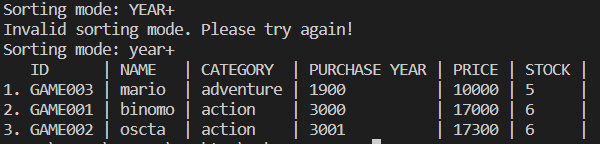
Input



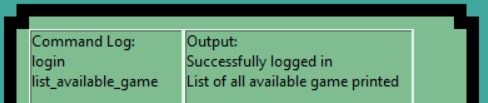
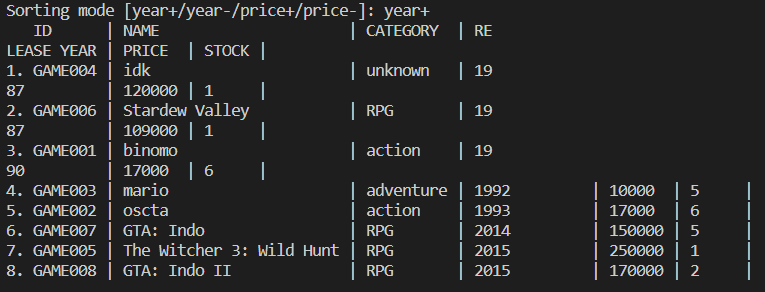
Output



Validasi



Implementasi di Main Program



**F08 –**

Input

Output

Validasi

Implementasi di Main Program

**F09 –**

Input

Output

Validasi

Implementasi di Main Program

**F10 – Mencari Game yang Dimiliki**

Input

Sebuah gambar berisi teks, jeruk, gelap

Deskripsi dibuat secara otomatis

Gambar 9.10.1 Input untuk modul F10

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

Gambar 9.10.2 Initial Database untuk pengecekan

Output

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

Gambar 9.10.3 Output untuk modul F10

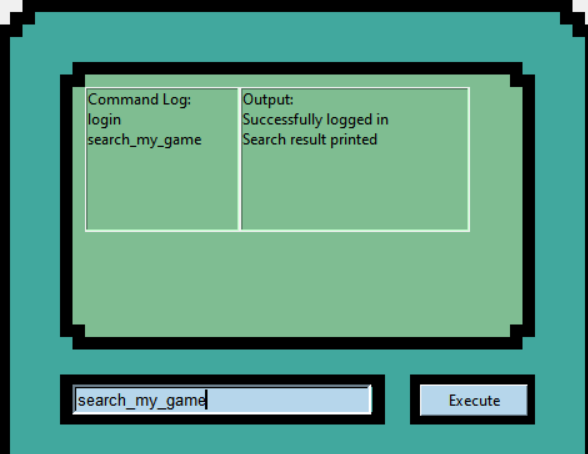
Validasi

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

Gambar 9.10.4 Validasi untuk modul F10

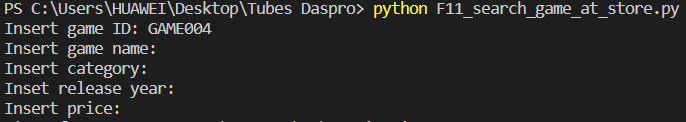
Implementasi di Main Program



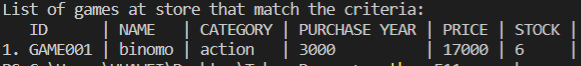
Gambar 9.10.5 Implementasi pemanggilan modul F10 pada main program

**F11 –**

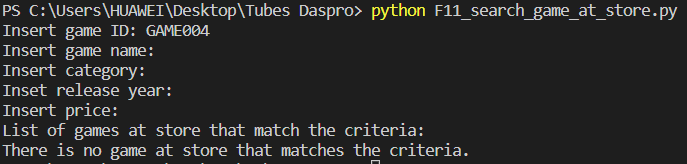
Input



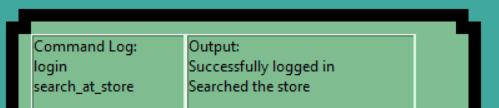
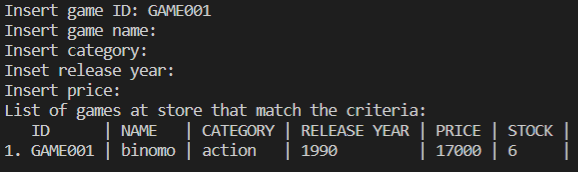
Output



Validasi

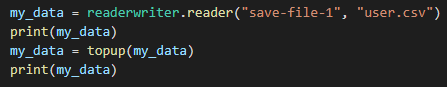


Implementasi di Main Program



**F12 – Top Up Saldo**

Input



Gambar 9.12.1 Cara memanggil prosedur dalam modul F12



Gambar 9.12.2 Input untuk modul F12

Output

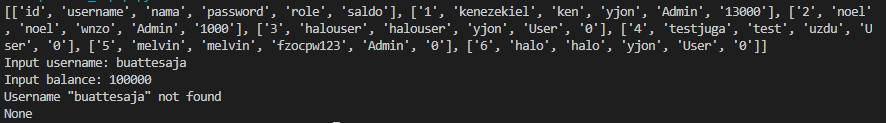


Gambar 9.12.3 Sebelum modifikasi register

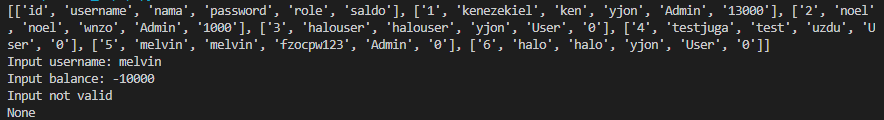


Gambar 9.12.4 Output untuk modul F12

Validasi

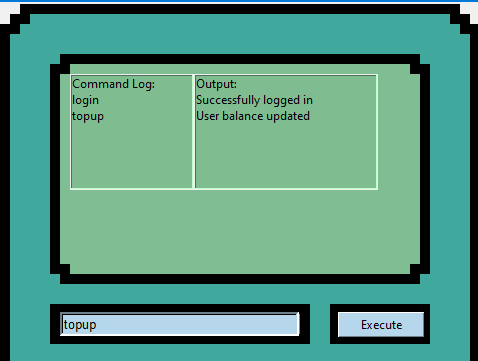


Gambar 9.12.5 Testing validasi username untuk input modul F12



Gambar 9.12.6 Testing validasi saldo untuk input modul F12

Implementasi di Main Program



Gambar 9.2.7 Contoh implementasi modul F12 pada main program

**F13 – Melihat Riwayat Pembelian**

**F14 – Help**

Input

Chart, treemap chart

Description automatically generated

Output

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Validasi

Text

Description automatically generated

Implementasi di Main Program

Chart, treemap chart

Description automatically generated

**F15 – Load**

Input



Output

A screenshot of a computer

Description automatically generated with medium confidence

Validasi

Text

Description automatically generated

Text

Description automatically generated

Implementasi di Main Program

Text

Description automatically generated

**F16 – Save**

Input

Graphical user interface, text

Description automatically generated

Output

A screenshot of a computer

Description automatically generated with medium confidence

Validasi

A picture containing text, orange, dark, close

Description automatically generated

Implementasi di Main Program

Diagram

Description automatically generated

**F17 – Exit**

Input

Output

Validasi

Implementasi di Main Program

**B01 – Cipher**

Input

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

Gambar 9.18.1 Cara memanggil fungsi dalam modul B01

Output

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

Gambar 9.18.2 Output untuk modul B01

Validasi

Tidak terdapat validasi karena diasumsikan semua password yang masuk telah tervalidasi oleh module F02 - Register

Implementasi di Main Program

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

Gambar 9.18.3 Hasil implementasi modul B01 pada database main program

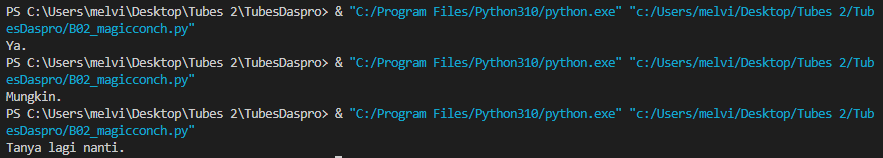
**B02 – Magic Conch**

Input



Gambar 9.19.1 Cara memanggil fungsi dalam modul B02

Output

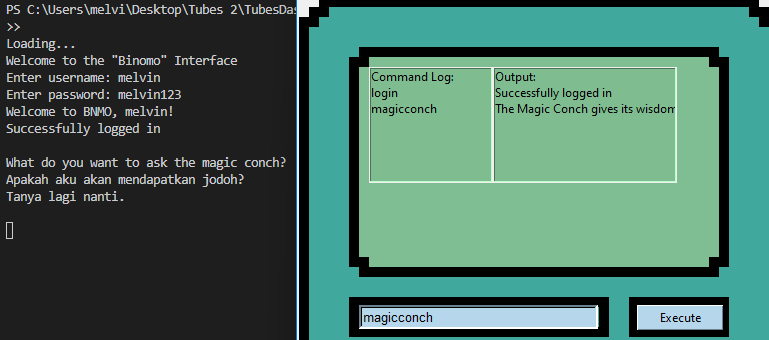


Gambar 9.19.2 Output untuk modul B02

Validasi

Tidak terdapat validasi karena function tidak menerima input apa pun.

Implementasi di Main Program



Gambar 9.19.3 Hasil implementasi modul B02 pada database main program

**B03 –**

Input

Output

Validasi

Implementasi di Main Program

Lampiran : Hasil Scan Forms Asitensi