DAFTAR PUSTAKA

Adams, E. (2013). *Fundamentals of Game Design* (3rd ed.). New Riders.

Asadi, A. (2016). *The Python Book : The Ultimate guide to coding with Python.* Bournemouth: Imagine Publishing Ltd.

Bachtiar, P. H., Wardhono, W. S., & Afirianto, T. (2018, Desember). Pendekatan MDA Framework Pada Pengembangan Permainan Baby Care. *Jurnal Teknologi Informasi dan Ilmu Komputer, 2*(12), 7408-7416.

Barokum, M. K., Amma, A. R., & Armin, A. P. (2018). Game Pembelajaran Rambu Lalu Lintas Berbasis Android. *KONVERGENSI, 14*(1), 11-17.

Barry, P. (2016). *Head First Python: A Brain-Friendly Guide* (2nd ed.). O'Reilly Media.

Bhoyarkar, A., Solanki, A., & Balbudhe, A. (2019). Application Development using Kivy Framework. *IJARCCE, 8*(2), 53-58.

Bootup Academy AI. (2019, May 27). *Android Adalah? Pengertian, Sejarah, Hingga Kelebihanya dibanding OS lain – BootUP.ai Blog*. Retrieved June 1, 2021, from Bootup: https://bootup.ai/blog/apa-itu-android-pengertian-kelebihan/

Cardle, P. J. (2017). *Android App Development in Android Studio: Java + Android Edition For Beginners* (1st ed.). Manchester Academic Publishers.

Cui, X., & Shi, H. (2011). A*-based Pathfinding in Modern Computer Games. *IJCSNS International Journal of Computer Science and Network Security*, *11*(1), 125-130.

Educative Answers Team. (2022, October 29). *What is the A* algorithm?* Retrieved May 31, 2021, from Educative: Interactive Courses for Software Developers: https://www.educative.io/answers/what-is-the-a-star-algorithm

Fazriawan, K. (2018, August 9). *Membangun Aplikasi Android dengan Python Kivy*. Retrieved May 31, 2021, from Codepolitan: https://www.codepolitan.com/membangun-aplikasi-android-dengan-python-kivy-5b61d26882da1

Galehantomo, G. (2015). Platform Comparison Between Games Console, Mobile Games And PC Games. *SISFORMA, 2*(1), 23-26.

Goyal, A., Mogha, P., Luthra, R., & Sangwan, N. (2014). Path Finding: A* or Dijkstra's? *International Journal in IT and Engineering, 2*(1), 1-15.

Griffiths, D., & Griffiths, D. (2017). *Head First Android Development: A Brain-Friendly Guide* (2nd ed.). O'Reilly Media.

Jointviews. (2021, February 18). *Mobile Game Development – It's Past, Present & Future*. Retrieved May 31, 2021, from Redbytes: https://www.redbytes.in/mobile-game-development-its-past-present-future/

Luthansyah, Y. D., Wardhono, W. S., & Brata, A. H. (2018, Oktober). Pengembangan Permainan Mobile AR Fishing Berbasis Marker Menggunakan Metode Iterative And Rapid Prototyping. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, 2*(10), 4255-4263.

Mathew, G. E. (2015). Direction Based Heuristic for Pathfinding in Video Games. *Procedia Computer Science, 47*, 262-271.

Phillips, D. (2014). *Creating Apps in Kivy: Mobile with Python* (1st ed.). O'Reilly Media.

Rahayu, D. P. (2020). Unit Testing Pada Aplikasi Web Mobile (Studi Kasus Bisnis Jasa Laundry). *Sarjana Thesis, Universitas Islam Indonesia*. DSpace UII. Retrieved from https://dspace.uii.ac.id/123456789/28341

Septiko, W. A. (2018). Pengembangan Game Edukasi Platformer Kisah Gajah Mada Menyatukan Nusantara Menggunakan Metode Iterative With Rapid Prototyping. *Sarjana Thesis, Universitas Brawijaya*. Repository UB. Retrieved from http://repository.ub.ac.id/id/eprint/162074

Setiawan, R. (2021, November 1). *Black Box Testing Untuk Menguji Perangkat Lunak*. Retrieved May 10, 2022, from Dicoding Blog: https://www.dicoding.com/blog/black-box-testing/

Snyder, C. (2003). *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces (Interactive Technologies)* (1st ed.). Morgan Kaufmann Publishers.

Tilawah, H. (2011). Penerapan Algoritma A-star (A*) Untuk Menyelesaikan Masalah Maze. *Makalah IF3051 Strategi Algoritma*.

LAMPIRAN

Lampiran 1 main.py

```
from math import sqrt
from random import randint, sample
from os.path import dirname
from functools import partial

from app.core import Player, CheckerBall

import kivy
import threading

kivy.require('1.0.0')

from kivy.config import Config
from kivy.modules import Modules
from kivy.clock import Clock
from kivy.core.window import Window
from kivy.utils import platform
from kivy.input.recorder import Recorder
from kivy.properties import ObjectProperty, NumericProperty

from kivy.uix.screenmanager import ScreenManager
from kivy.uix.gridlayout import GridLayout
from kivy.uix.button import Button

from kivymd.app import MDApp
from kivymd.toast import toast
from kivymd.uix.screen import MDScreen
from kivymd.uix.button import MDFlatButton
from kivymd.uix.dialog import MDDialog

game_difficulty = ObjectProperty(None)
Modules.add_path(dirname(__file__))
Config.set("modules", "telenium_client", "")

# StartScreen
class StartScreen(MDScreen):
    pass

# GameScreen
class GameScreen(MDScreen):
    players = {}
```

```
dynamic_value = 10
grid = []
free_cells = []
set_balls = []
selected = ''
is_game_over = False
destroyed = False
prev_ball = ''
prev_color = ''
difficulty = ''
layout = ''
board_layout = ''
turn = 'pl'
score = 0
clicked = False
dialog = None
game_dialog = None
value_max = 0
max_ball = 2
second = -1
minute = 0
score_text = ObjectProperty(None)

step_counter = NumericProperty(0)
#game_time = ObjectProperty(None)
game_score = ObjectProperty(None)

def show_success_toast(self):
    toast('movement success.')

def show_selected_toast(self, row, column):
    toast('ball selected at ['+str(row)+'] ['+str(column)+']')

def __init__(self, **kwargs):
    super(GameScreen, self).__init__(**kwargs)
    self.coordinates = None
    self.set_balls = []
    self.event = Clock.schedule_once(self._finish_init)

def _finish_init(self, dt):
    self.layout = self.ids.board_one
    self.dynamic_value = 10
    self.free_cells = []
    grid_value = self.dynamic_value * self.dynamic_value
    diviser = int(sqrt(grid_value))
    self.board_layout = GridLayout(cols = self.dynamic_value, rows = self.dynamic_value)
```

```
    nbBalls = int((self.dynamic_value / 2) * ((self.dynamic_value / 2 )- 1))
    self.players['pl'] = Player(name = 'Player', player = 'pl', score = nbBalls)
    self.value_max = self.max_color(self.game_difficulty.text)
    random_balls = sample(range(0, grid_value), 3)
    ordering = sorted(random_balls)

    for i in range(0, grid_value):
        try:
            value = self.grid[int(i / (diviser * (i % diviser)))]
        except Exception as e:
            self.grid.append([])

        button_id = f"{str(int(i/diviser))}-{str(int(i%diviser))}"

        y = int(i / diviser)
        x = int(i % diviser)

        self.grid[y].append([])
        self.grid[y][x].append('')
        self.free_cells.append((x, y))

        if i in ordering:
            ball = 'pl'
            color = randint(1, self.value_max)
        else:
            ball = ''
            color = 0

        button = Button(on_release=self.move_object)
        button.id = button_id
        self.draw_tiles(button, color)
        self.grid[y][x] = CheckerBall(ball, color, button, y, x)
        if self.grid[y][x].ball == 'pl':
            self.free_cells.remove((x, y))
            self.set_balls.append((x, y))
        self.board_layout.add_widget(self.grid[y][x].button)
    self.layout.add_widget(self.board_layout)
    self.start_time()

# def start_time(self):
#    Clock.unschedule(self.time_increment)
#    Clock.schedule_interval(self.time_increment, 1)

# def time_increment(self, dt):
#    self.second += 1
    if self.second == 60:
```

```
            self.second = 0
            self.minute += 1
        self.game_time.text = ('%02d:%02d'%(self.minute, self.second))

    # def stop_time(self):
    #    if self.minute > 0:
    #        self.minute = 0
    #    self.second = -1


    def add_balls(self):
        self.value_max = self.max_color(self.game_difficulty.text)
        for i in range(self.max_ball):
            if len(self.free_cells) > self.max_ball:
                coord = self.free_cells[randint(0, len(self.free_cells)) - 1]
                self.grid[coord[1]][coord[0]].ball = 'pl'
                self.grid[coord[1]][coord[0]].color = randint(1, self.value_max)
                self.grid[coord[1]][coord[0]].button.background_normal =
self.ball_color(self.grid[coord[1]][coord[0]].color)
                self.free_cells.remove((coord[0], coord[1]))
            else:
                self.is_game_over = True
                raise FieldFullException()

    def clear_board(self):
        self.free_cells.clear()
        for tile in range(0, int(self.dynamic_value * self.dynamic_value)):
            row = int(tile / self.dynamic_value)
            column = int(tile % self.dynamic_value)
            self.grid[row][column].ball = ''
            self.grid[row][column].color = 0
            self.grid[row][column].button.background_normal =
self.ball_color(self.grid[row][column].color)
            self.free_cells.append((row, column))


    def reset_board(self):
        self.clear_board()
        self.stop_time()
        self.score = 0
        self.game_score.text = str(self.score)
        self.add_balls()


    #Finding line ball
    def find_lines(self, row, column):
```

```
        if self.grid[row][column].color == 0:
            return

        current_color = self.grid[row][column].color
        delete_ball = []

        minus_dx = column
        plus_dx = column + 1
        while minus_dx >= 0 and self.grid[row][minus_dx].color == current_color:
            delete_ball.append((minus_dx, row))
            minus_dx -= 1
        while plus_dx < self.dynamic_value and self.grid[row][plus_dx].color == current_color:
            delete_ball.append((plus_dx, row))
            plus_dx += 1
        if len(delete_ball) >= 5:
            return delete_ball
        else:
            delete_ball.clear()

        minus_dy = row
        plus_dy = row + 1
        while minus_dy >= 0 and self.grid[minus_dy][column].color == current_color:
            delete_ball.append((column, minus_dy))
            minus_dy -= 1
        while plus_dy < self.dynamic_value and self.grid[plus_dy][column].color ==
current_color:
            delete_ball.append((column, plus_dy))
            plus_dy += 1
        if len(delete_ball) >= 5:
            return delete_ball
        else:
            delete_ball.clear()

        minus_dx = column
        minus_dy = row
        plus_dx = column + 1
        plus_dy = row + 1
        while minus_dx >= 0 and minus_dy >= 0 and self.grid[minus_dy][minus_dx].color ==
current_color:
            delete_ball.append((minus_dx, minus_dy))
            minus_dx -= 1
            minus_dy -= 1
        while plus_dx < self.dynamic_value and plus_dy < self.dynamic_value and
self.grid[plus_dy][plus_dx].color == current_color:
            delete_ball.append((plus_dx, plus_dy))
            plus_dx += 1
```

```
            plus_dy += 1
        if len(delete_ball) >= 5:
            return delete_ball
        else:
            delete_ball.clear()

        minus_dx = column
        plus_dy = row
        while minus_dx >= 0 and plus_dy < self.dynamic_value and
self.grid[plus_dy][minus_dx].color == current_color:
            delete_ball.append((minus_dx, plus_dy))
            minus_dx -= 1
            plus_dy += 1

        plus_dx = column + 1
        minus_dy = row - 1
        while plus_dx < self.dynamic_value and minus_dy >= 0 and
self.grid[minus_dy][plus_dx].color == current_color:
            delete_ball.append((plus_dx, minus_dy))
            plus_dx += 1
            minus_dy -= 1
        if len(delete_ball) >= 5:
            return delete_ball
        else:
            return

    #deleting line ball
    def delete_line(self, ball_coord_list):
        if ball_coord_list is not None:
            self.scoring(len(ball_coord_list))
            for coord in ball_coord_list:
                self.grid[coord[1]][coord[0]].ball = ''
                self.grid[coord[1]][coord[0]].color = 0
                self.grid[coord[1]][coord[0]].button.background_normal =
self.ball_color(self.grid[coord[1]][coord[0]].color)
                self.free_cells.append((coord[0], coord[1]))

    # changing ball color
    def ball_color(self, color):
        colors = ('empty','aqua', 'black', 'blue',
                'dark_green', 'light_green', 'orange', 'pink',
                'red', 'yellow')
        color_list = {i: color for i, color in enumerate(colors)}
        color_ball = f"app/resources/img/{color_list[color]}.png"

        return color_ball
```

```
    #Drawing tiles
    def draw_tiles(self, button, color):
        button.background_normal = self.ball_color(color)
        button.size = self.size
        button.border = (0,0,0,0)

    #moving object using A* algorithm
    def moving(self, start_x, start_y, end_x, end_y):
        if (self.grid[end_y][end_x].ball != '' and self.grid[end_y][end_x].color != 0) or
(self.grid[start_y][start_x].ball == '' and self.grid[start_y][start_x].color == 0):
            return False
        queue = [] #open list
        visited = [] #closed list
        queue.append((start_x, start_y)) #add x,y node to open list
        while len(queue) != 0:
            coord = queue.pop(0) #remove x,y node from open list
            if coord[0] < 0 or coord[0] >= self.dynamic_value or coord[1] < 0 or coord[1] >=
self.dynamic_value:
                continue
            if (coord != (start_x, start_y) and (self.grid[coord[1]][coord[0]].ball != '' and
self.grid[coord[1]][coord[0]].color != 0)) or (coord[0], coord[1]) in visited: #check if the
node path is available
                continue
            if coord[0] == end_x and coord[1] == end_y: #current == goal ?
                return True
            visited.append((coord[0], coord[1]))
            for dx in range(-1, 2):
                for dy in range(-1, 2):
                    if dx != 0 and dy != 0:
                        continue
                    else:
                        queue.append((coord[0] + dx, coord[1] + dy))
        return False

    # move the object
    def move_object(self, button):
        try:
            matching = button.id.split('-')
            row = int(matching[0])
            column = int(matching[1])
            ball = self.grid[row][column]

            if column < 0 or column >= self.dynamic_value or column < 0 or column >=
self.dynamic_value:
                return
```

```
        if self.clicked == True:
            self.show_selected_toast(row, column)
            if row == self.prev_ball.row and column == self.prev_ball.column:
                self.reinit_prev()
                self.clicked = False
            elif ball.ball == '':
                if self.prev_ball.ball != '':
                    if self.moving(self.prev_ball.column, self.prev_ball.row, column, row):
                        self.thread = threading.Thread(
                                        target=self.move_ball,
                                        args=(self.prev_ball.row,
                                              self.prev_ball.column,
                                              row, column))
                        threading.Event().wait(0.2)
                        self.thread.start()
                    self.prev_ball = ''
                    self.show_success_toast()
            elif ball.ball != '':
                self.prev_ball.button.background_color = (1, 1, 1, 1)
                self.prev_ball = ball
                self.clicked = True
                ball.button.background_color = (1, 1, 1, 0.7)

            else:
                return False
        else:
            if ball.ball == self.turn:
                self.prev_ball = ball
                ball.button.background_color = (1, 1, 1, 0.7)
                self.clicked = True
        return False
    except FieldFullException:
        print(f'game over. Score : {self.score}')
        self.game_over_dialog()

def max_color(self, difficulty):

    max_number = 3

    if difficulty == "Easy":
        max_number = 3
    elif difficulty == "Medium":
        max_number = 6
    elif difficulty == "Hard":
        max_number = 9
```

```
        return max_number

    def place(self, prev_row, prev_column, row, column):
        if self.prev_ball.ball == self.turn:
            self.check(prev_row, prev_column, row, column)
            return False
        return False

    # checking grid
    def check(self, prev_row, prev_column, row, column):
        if self.check_moveable(prev_row, prev_column, row, column) == True:
            if self.destroyed == False:
                self.change_turn()
        return False

    # moving check
    def check_moveable(self, prev_row, prev_column, row, column):
        if self.grid[row][column].ball == '' and self.grid[row][column].color == 0:
            return self.swap_ball_values(prev_row, prev_column, row, column)
        return False

    def move_ball(self, prev_row, prev_column, row, column):
        self.place(prev_row, prev_column, row, column)
        find_lines = self.find_lines(row, column)
        if find_lines is None:
            self.add_balls()
            for coordinates in self.set_balls:
                array = self.find_lines(coordinates[0], coordinates[1])
                if array is not None:
                    self.delete_line(array)
        else:
            self.delete_line(find_lines)
    # def update(self, *args):
    #     self.prev_ball

    def reinit_prev(self):
        self.grid[self.prev_ball.row][self.prev_ball.column].button.background_normal =
self.ball_color(self.prev_ball.color)
        self.prev_ball = 0

    def swap_ball_values(self, prev_row, prev_column, row, column):
        self.grid[row][column].ball, self.grid[prev_row][prev_column].ball =
self.grid[prev_row][prev_column].ball, self.grid[row][column].ball
        self.grid[row][column].color, self.grid[prev_row][prev_column].color =
self.grid[prev_row][prev_column].color, self.grid[row][column].color
```

```
        self.free_cells.remove((column, row))
        self.free_cells.append((prev_column, prev_row))
        self.grid[row][column].button.background_normal =
self.ball_color(self.grid[row][column].color)
        self.destroyed = False
        self.reinit_prev()
        return True

    def change_turn(self):
        self.turn = self.players[self.turn].player
        self.clicked = False
        return False

    def game_over(self):
        self.is_game_over = True

    def restart(self, *args):
        self.reset_board()
        self.dialog.dismiss()

    def restart_game(self, *args):
        self.reset_board()
        self.game_dialog.dismiss()

    def scoring(self, length_of_remote_line):
        multiplier = length_of_remote_line % 5+1
        self.score += 10 * length_of_remote_line * multiplier
        self.game_score.text = str(self.score)

    def set_screen(self):
        MDApp.get_running_app().root.current = "start"
        MDApp.get_running_app().root.transition.direction = "right"

    def return_screen(self, *args):
        MDApp.get_running_app().root.current = "start"
        MDApp.get_running_app().root.transition.direction = "right"
        self.restart()

    def on_pre_enter(self):
        self.reset_board()

    def game_over_dialog(self):
        if not self.game_dialog:
            self.game_dialog = MDDialog(
                    title = "Game Over", text = "your journey ends, Do you want to restart ?",
                    buttons = [
```

```
                    MDFlatButton(
                        text="YES", text_color=
MDApp.get_running_app().theme_cls.primary_color, on_release = self.restart_game
                    ),
                    MDFlatButton(
                        text="NO", text_color= MDApp.get_running_app().theme_cls.primary_color,
on_release = self.close_game_dialog
                    )
                ]
            )
        self.game_dialog.open()

    def back_dialog(self):
        if not self.dialog:
            self.dialog = MDDialog(
                    title = "Return to Menu", text = "do you want to return to menu ?",
                    buttons = [
                    MDFlatButton(
                        text="YES", text_color=
MDApp.get_running_app().theme_cls.primary_color, on_release = self.return_screen
                    ),
                    MDFlatButton(
                        text="NO", text_color= MDApp.get_running_app().theme_cls.primary_color,
on_release = self.close_dialog
                    )
                ]
            )
        self.dialog.open()

    def close_dialog(self, inst):
        self.dialog.dismiss()

    def close_game_dialog(self, inst):
        self.game_dialog.dismiss()


class FieldFullException(Exception):
    """Field full and no places to set next balls"""
    pass

# SettingsScreen
class SettingsScreen(MDScreen):
    def __init__(self, **kwargs):
        super(SettingsScreen, self).__init__(**kwargs)

    def set_screen(self):
```

```
        MDApp.get_running_app().root.current = "start"
        MDApp.get_running_app().root.transition.direction = "right"

    def on_checkbox_active(self, instance, value, difficulty):
        if value == True:
            game_screen = MDApp.get_running_app().root.get_screen("game")
            game_screen.game_difficulty.text = difficulty

    def on_max_ball_checkbox_active(self, instance, value, ball_max):
        if value == True:
            game_screen = MDApp.get_running_app().root.get_screen("game")
            game_screen.max_ball = int(ball_max)

# HelpScreen
class HelpScreen(MDScreen):
    def set_screen(self):
        MDApp.get_running_app().root.current = "start"
        MDApp.get_running_app().root.transition.direction = "right"

# RootScreen
class RootScreen(ScreenManager):
    pass

# Main
class MainApp(MDApp):
    dialog = None

    def build(self):
        Config.set('graphics', 'width', 360)
        Config.set('graphics', 'height', 720)
        Config.set('graphics', 'resizable', 0)
        Config.set('graphics', 'borderless', 1)
        # Config.set('input', 'mouse', 'mouse,disable_multitouch')
        Config.set('input', 'mouse', 'mouse,multitouch_on_demand')
        Config.write()
        if platform not in ["android", "ios"]:
            Window.size = (360,720)
        Window.borderless = False
        Clock.max_iteration = 20
        self.title = "Magic Line"

        return RootScreen()

    def show_exit_dialog(self):
        if not self.dialog:
            self.dialog = MDDialog(
```

```
                title ="Exit",
                text = "Are You Sure you want to exit ?",
                buttons = [
                    MDFlatButton(
                        text="YES", text_color= self.theme_cls.primary_color, on_release =
self.stop
                    ),
                    MDFlatButton(
                        text="NO", text_color= self.theme_cls.primary_color, on_release =
self.close_dialog
                    )
                ]
            )
        self.dialog.open()

    def close_dialog(self, inst):
        self.dialog.dismiss()


if __name__ =="__main__":
    MainApp().run()
```

Lampiran 2 main.kv

```
#:kivy 2.1.0
#:import SlideTransition kivy.uix.screenmanager.SlideTransition
#:import BackgroundColorBehavior kivymd.uix.behaviors.backgroundcolor_behavior

<Check@MDCheckbox>:
    group: 'group'
    size_hint: None, None
    size: dp(48), dp(48)

<Check_two@MDCheckbox>:
    group: 'group_2'
    size_hint: None, None
    size: dp(48), dp(48)

<RootScreen>
    transition: SlideTransition()
    StartScreen:
    GameScreen:
    SettingsScreen:
    HelpScreen:

<StartScreen>
    name: "start"
    FitImage:
        source: 'app/resources/img/bg.png'

    MDBoxLayout:
        orientation:'vertical'

        AnchorLayout:
            size_hint: 1, 0.4
            anchor_x: 'center'
            anchor_y: 'top'
            padding: 25, 50, 25, 25

            MDBoxLayout:
                orientation: 'vertical'

                MDLabel:
                    text: "Magic Line"
                    halign: "center"
                    font_style: 'H3'

        AnchorLayout:
```

```
size_hint: 1, 0.6
anchor_x: 'center'
anchor_y: 'bottom'
padding: 25, 25, 25, 50

MDBoxLayout:
    orientation: 'vertical'
    spacing: 1

    MDFillRoundFlatButton:
        text: "Start"
        md_bg_color: 0, 0, 0, 1
        on_release:
            root.manager.transition.direction = 'left'
            root.manager.transition.duration = 0.1
            root.manager.current = "game"
        size: self.width, dp(50)
        size_hint: .75, None
        pos_hint: {'center_x':.5, 'center_y':.5}

    MDFillRoundFlatButton:
        text: "Settings"
        md_bg_color: 0, 0, 0, 1
        on_press:
            root.manager.transition.direction = 'left'
            root.manager.transition.duration = 0.1
            root.manager.current = "settings"
        size: self.width, dp(50)
        size_hint: .75, None
        pos_hint: {'center_x':.5, 'center_y':.5}

    MDFillRoundFlatButton:
        text: "Help"
        md_bg_color: 0, 0, 0, 1
        on_release:
            root.manager.transition.direction = 'left'
            root.manager.transition.duration = 0.1
            root.manager.current = "help"
        size: self.width, dp(50)
        size_hint: .75, None
        pos_hint: {'center_x':.5, 'center_y':.5}

    MDFillRoundFlatButton:
        text: "Exit"
        md_bg_color: 0, 0, 0, 1
        on_release:
```

```
                        app.show_exit_dialog()
                    size: self.width, dp(50)
                    size_hint: .75, None
                    pos_hint: {'center_x':.5, 'center_y':.5}

<GameScreen>
    name:"game"
    game_difficulty: game_difficulty
    game_score: game_score
    game_time: game_time
    FitImage:
        source: 'app/resources/img/bg.png'

    MDBoxLayout:
        orientation:'vertical'

        MDTopAppBar:
            title: "Magic Line"
            size_hint_y: 0.1
            left_action_items: [["arrow-left", lambda x: root.back_dialog()]]

        MDBoxLayout:
            orientation: 'horizontal'
            size_hint_y: 0.15

            MDBoxLayout:
                orientation: 'vertical'
                md_bg_color: 1, 1, 1, 1

                MDLabel:
                    text: "Difficulty"
                    halign: 'center'

                MDLabel:
                    id: game_difficulty
                    halign: 'center'
                    text: 'Easy'

       #     MDBoxLayout:
       #         orientation: 'vertical'
       #         md_bg_color: 1, 1, 1, 1

       #         MDLabel:
       #             text: "Time"
       #             halign: 'center'
```

```
        #        MDLabel:
        #            id: game_time
        #            text: '00:00'
        #            halign: 'center'

            MDBoxLayout:
                orientation: 'vertical'
                md_bg_color: 1, 1, 1, 1

                MDLabel:
                    text: "Score"
                    halign: 'center'

                MDLabel:
                    id: game_score
                    text: '0'
                    halign: 'center'

        MDBoxLayout:
            orientation: 'horizontal'
            size_hint_y: 0.75

            RelativeLayout:
                id: board_one
                size_hint: None, None
                size: root.width, root.width
                pos_hint: {'center_x':0.5, 'center_y':0.5}

                Image:
                    source: 'app/resources/img/Board2.png'
                    allow_stretch: False
                    keep_ratio: True

<SettingsScreen>
    name:"settings"

    FitImage:
        source: 'app/resources/img/bg.png'

    MDBoxLayout:
        orientation: "vertical"

        AnchorLayout:
            size_hint: 1, 0.25
            anchor_x: 'center'
            anchor_y: 'top'
```

```
            MDTopAppBar:
                title: "Settings"
                left_action_items: [["arrow-left", lambda x: root.set_screen()]]

        AnchorLayout:
            size_hint: 1, 0.75
            anchor_x: 'center'
            anchor_y: 'center'
            padding: 25, 0, 25, 75

            MDGridLayout:
                md_bg_color: 1, 1, 1, 1
                padding: 25, 0, 25, 25
                cols: 2

                MDLabel:
                    text:"Difficulty"
                    halign: 'center'

                MDBoxLayout:
                    orientation: 'vertical'

                    MDGridLayout:
                        cols: 2

                        Check:
                            id: easy
                            size_hint: 1, 0.2
                            size: "24dp", "28dp"
                            on_active: root.on_checkbox_active(self, self.active,
root.ids.lbl_easy.text)
                            active: True
                            allow_no_selection: False

                        MDLabel:
                            id: lbl_easy
                            text: "Easy"
                            font_size: '16sp'

                        Check:
                            id: medium
                            size_hint: 1, 0.2
                            size: "24dp", "28dp"
                            on_active: root.on_checkbox_active(self, self.active,
root.ids.lbl_medium.text)
```

```
                            allow_no_selection: False

                        MDLabel:
                            id: lbl_medium
                            text: "Medium"
                            font_size: '16sp'

                        Check:
                            id: hard
                            size_hint: 1, 0.2
                            size: "24dp", "28dp"
                            on_active: root.on_checkbox_active(self, self.active,
root.ids.lbl_hard.text)
                            allow_no_selection: False

                        MDLabel:
                            id: lbl_hard
                            text:"Hard"
                            font_size: '16sp'

                MDLabel:
                    text:'Max Ball'
                    halign: 'center'
                    md_bg_color: (242/255), (238/255), (7/255), 0.61

                MDBoxLayout:
                    orientation: 'vertical'
                    md_bg_color: (242/255), (238/255), (7/255), 0.61

                    MDGridLayout:
                        cols: 2

                        Check_two:
                            id: two
                            size_hint: 1, 0.2
                            size: "24dp", "28dp"
                            on_active: root.on_max_ball_checkbox_active(self, self.active,
root.ids.lbl_two.text)
                            active: True
                            allow_no_selection: False

                        MDLabel:
                            id: lbl_two
                            text: "2"
                            font_size: '16sp'
```

```
                        Check_two:
                            id: three
                            size_hint: 1, 0.2
                            size: "24dp", "28dp"
                            on_active: root.on_max_ball_checkbox_active(self, self.active,
root.ids.lbl_three.text)
                            allow_no_selection: False

                        MDLabel:
                            id: lbl_three
                            text:"3"
                            font_size: '16sp'
<HelpScreen>
    name:"help"
    FitImage:
        source: 'app/resources/img/bg.png'

    MDBoxLayout:
        orientation: "vertical"

        AnchorLayout:
            size_hint: 1, 0.25
            anchor_x: 'center'
            anchor_y: 'top'

            MDTopAppBar:
                title: "Help"
                left_action_items: [["arrow-left", lambda x: root.set_screen()]]

        AnchorLayout:
            size_hint: 1, 0.75
            anchor_x: 'center'
            anchor_y: 'top'
            padding: 25, 0, 25, 75

            MDBoxLayout:
                md_bg_color: 1, 1, 1, 1
                orientation: 'vertical'
                padding: 25, 0, 25, 25

                MDLabel:
                    size_hint: 1, 0.38
                    text: "How to Play"
                    halign: "center"
                    font_style: 'H6'
```

```
                MDLabel:
                    text: "Pemain ditugaskan untuk membuat garis yang terdiri dari
sekumpulan bola berwarna sejenis sebanyak lima buah atau lebih untuk mendapatkan poin.
Namun bola yang akan dipindahkan harus melalui jalur kosong yang tidak terhalang oleh
bola-bola lainnya, atau bola yang  dipilih tersebut tidak dapat bergerak. Makin cepat
Pemain membersihkan papan permainan maka makin banyak poin yang bisa didapat dan makin
lama pula pemain dapat bertahan dalam permainan tersebut."
                    halign: "justify"
                    font_style: 'Body1'
```

Lampiran 3 core.py

```
from random import randint
class CheckerBall:

    def __init__(self, ball = '', color = 0, button = 0, row = 0, column = 0):
        self.ball = ball
        self.color = color
        self.button = button
        self.row = row
        self.column = column
        self.clicked = False
        return

class Player:
    name = ''
    score = 0

    def __init__(self, name, player, score):
        self.name = name
        self.player = player
        self.score = score
        return
```