

3, Menara Sentral, Penang Street, 10200 Penang. Tel: 04-2621888

ASSIGNMENT COVER SHEET

|  |
| --- |
| Student Name: Jaden Teh Ee Zhe, Tan Da Wei |
| Student ID: SC08250078DC, SC08250082DC |
| Subject (Title + Code): Introduction to Programming (EIP3303) |
| Semester / Year: August 2025 |
| Lecturer: Ms. Navitha |
| Lecturer’s Comments: |

(*This completed cover sheet must be attached to the front of your assignment)*

Plagiarism

Plagiarism is the misrepresentation of the work of others as one’s own (including ideas, arguments, words, diagrams, images or data). It includes the explicit claim that another person’s work as one’s own and the failure to adequately acknowledge the sources used. Plagiarism is an academic misconduct and will be penalized accordingly.

Declaration

I certify that this assignment is a result of my own independent work and investigation. The use of material from other sources has been properly and fully acknowledged in this assignment. A bibliography is appended. I fully understand the consequence of committing plagiarism, and if proven and in the absence if mitigating circumstances, it could include failure in the Year or part of my course or even withdrawal from the program.

I declare that this assignment is in full compliance of the above requirements.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Signature: | Jaden，Tan |  | Date: | 12/11/2025 |

* To score high marks, you must meet the requirements outlined in the assessment criteria. It is attached to your assignment question paper.
* Late submission might result in mark deduction of marks.

Source Code:

# ======================================================================================================================  
# 🐍🪜 SNAKES AND LADDER GAME (By Jaden & David)  
# ======================================================================================================================  
  
# This module is a Random Variable Generator  
# That allows us to implement randomizing algorithm systems  
# in python  
import random  
  
# Defines Ladders value for the game.  
# Structure -> (startPosition: endPosition)  
LADDERS = {  
 3: 20, 6: 14, 11: 28, 15: 34, 17: 74, 22: 37,  
 38: 59, 49: 67, 57: 76, 61: 78, 73: 86, 81: 98, 88: 91  
}  
  
# Defines Snakes value for the game.  
# Structure -> (snakeHead: snakeTail)  
SNAKES = {  
 8: 4, 18: 1, 26: 10, 39: 5, 51: 6, 54: 36, 56: 40,  
 60: 23, 75: 28, 83: 63, 85: 59, 90: 48, 92: 25, 97: 87, 99: 79  
}  
  
gameResults = []  
players = [  
 ["P01", "Player1", 0],  
 ["P02", "Player2", 0]  
]  
  
class DataHandler:  
  
 def \_\_init\_\_(self):  
 self.playerIndex = 0  
  
 def registerUser(self):  
  
 # Capture all the existing players' IDs and Names.  
 existingIds = [player[0] for player in players]  
 existingNames = [player[1] for player in players]  
  
 # Make sure it runs indefinitely  
 while True:  
  
 # Ask for Player's ID.  
 print()  
 print("Enter your Player ID (5 - 20 char) to register")  
 playerId = input("or type 'e' to exit: ")  
  
 # Let player has the authority to exit anytime they like to.  
 if playerId == 'e' or playerId == 'E':  
 break  
  
 # Check if duplication exists in the database.  
 elif playerId in existingIds:  
 print("❌ This Player ID is already registered.")  
 break  
  
 # Prevent empty player ID.  
 elif playerId.isspace() or playerId == "":  
 print("❗ Player ID CANNOT be empty!")  
 break  
  
 # Prevent short or long player ID.  
 elif len(playerId) > 20:  
 print("❗ Player ID must be LESSER than 20 characters!")  
 break  
 elif len(playerId) < 5:  
 print("❗ Player ID must be LARGER than 5 characters!")  
 break  
  
 # Ask for Player's name.  
 playerName = input("Enter your Player Name or type 'e' to exit: ")  
  
 # Let player has the authority to exit anytime they like to.  
 if playerName == 'e' or playerName == 'E':  
 break  
  
 # Check if duplication exists in the database.  
 elif playerName in existingNames:  
 print("❌ This Player Name is already registered.")  
 break  
  
 # Starts to run the process if all the requirements are passed.  
 else:  
 newPlayerData = [playerId, playerName, 0]  
 players.append(newPlayerData)  
 print(f"✅ Player ID: {playerId} | Player Name: {playerName} has been registered")  
  
 # To let the player able to read and understand all the infos or errors,  
 # An input() function is placed to prevent the code in the next part runs.  
 # The Player can then press <Enter> to proceed on the next part.  
 print()  
 input("Press <Enter> to continue...")  
 print()  
  
 def editPlayersInfo(self):  
  
 # Edit an existing player's name using their player ID.  
  
 # Capture all the existing players' IDs and Names.  
 existingIds = [player[0] for player in players]  
 existingNames = [player[1] for player in players]  
  
 # Make sure it runs indefinitely  
 while True:  
  
 # Ask for Player's ID.  
 print()  
 print("Enter your Player ID to edit info")  
 playerId = input("or type 'e' to exit: ")  
  
 # Let player has the authority to exit anytime they like to.  
 if playerId == 'e' or playerId == 'E':  
 break  
  
 # Prevent empty player ID.  
 elif playerId.isspace() or playerId == "":  
 print("❗ Player ID CANNOT be empty!")  
 break  
  
 # If the player's ID does not exist,  
 # This code will run to inform typo and other possible errors.  
 elif playerId not in existingIds:  
 print("❓ This Player ID is not registered.")  
 break  
  
 # Ask for Player's name.  
 newPlayerName = input("Input your new Player Name or type 'e' to exit: ")  
  
 # Let player has the authority to exit anytime they like to.  
 if newPlayerName == 'e' or newPlayerName == 'E':  
 break  
  
 # Check if duplication exists in the database.  
 elif newPlayerName in existingNames:  
 print("❌ This Player Name is already existed.")  
 break  
  
 # Starts to run the process if all the requirements are passed.  
 else:  
 for player in players:  
 if playerId == player[0]:  
 player[1] = newPlayerName  
 self.playerIndex = 0  
 break  
 else:  
 self.playerIndex += 1  
  
 print(f"✅ Player Name: {newPlayerName} has been updated")  
  
 # To let the player able to read and understand all the infos or errors,  
 # An input() function is placed to prevent the code in the next part runs.  
 # The Player can then press <Enter> to proceed on the next part.  
 print()  
 input("Press <Enter> to continue...")  
 print()  
  
 def deletePlayers(self):  
  
 # Delete a player infos and record from the database by player ID.  
  
 # Capture all the existing players' IDs.  
 existingIds = [player[0] for player in players]  
  
 # Make sure it runs indefinitely  
 while True:  
  
 # Ask for player's ID  
 print()  
 print("Enter the Player ID to delete")  
 playerId = input("or type 'e' to exit: ")  
  
 # Let player has the authority to exit anytime they like to.  
 if playerId == 'e' or playerId == 'E':  
 break  
  
 # Prevent empty player ID.  
 elif playerId.isspace() or playerId == "":  
 print("❌ Player ID CANNOT be empty!")  
  
 # If the player's ID does not exist,  
 # This code will run to inform typo and other possible errors.  
 elif playerId not in existingIds:  
 print("❓ This Player ID is not registered.")  
  
 # Starts to run the process if all the requirements are passed.  
 else:  
  
 # Using for loops to find player's info  
 # by player's ID.  
 for player in players:  
  
 # If the player's ID is found in the  
 # "players" list, the program will stop.  
 if playerId == player[0]:  
 break  
  
 # Else, the variable that indicates  
 # player's index will be incremented by 1.  
 else:  
 self.playerIndex += 1  
  
 # Starts to run the process if all the requirements are passed.  
 print(self.playerIndex)  
 players.pop(self.playerIndex)  
 self.playerIndex = 0  
 print(f"🗑️ Player ID: {playerId} has been deleted")  
  
 # To let the player able to read and understand all the infos or errors,  
 # An input() function is placed to prevent the code in the next part runs.  
 # The Player can then press <Enter> to proceed on the next part.  
 print()  
 input("Press <Enter> to continue...")  
 print()  
  
 def addGameResults(self, roundCount, winner, runnerUp, secondRunnerUp):  
 # Check if there are any game results exists  
 # as it will affect the naming of the Game ID.  
 if gameResults == []:  
 largestId = 0  
 else:  
  
 # result[0] represents the game ID  
 # The game ID format is "G001"  
 #  
 # Assume that "G001" is our game's ID.  
 # result[0] will be "G001"  
 # [1:] will chop its value,  
 # which lets it start from the second value to the end.  
 #  
 # " G 0 0 1 "  
 # ^ ^  
 # Start End  
 #  
 # result[0][1:] = "001"  
 # Conclusion: int(result[0][1:]) = 1  
 resultInInt = [int(result[0][1:]) for result in gameResults]  
  
 # max() will returns the maximum value of the list.  
 largestId = max(resultInInt)  
  
 # This is an increment function that will add up  
 # by one from the largest ID number.  
 #  
 # About ':03d' format, refers to the python text book (pg. 96 & 97)  
 #  
 # Example 1:  
 # print(f"{31:4d}")  
 # Output:  
 # 31  
 # It adds (4 - len("31")) empty spaces in front.  
 #  
 # Example 2:  
 # print(f"{31:04d}")  
 # Output:  
 # 000031  
 # It adds (4 - len("31")) zeros in front.  
 newGameId = f"G{largestId + 1:03d}"  
  
 # Starts to run this process if there are 3 or more players.  
 if secondRunnerUp != "":  
 newGameResult = [newGameId, roundCount, winner, runnerUp, secondRunnerUp]  
 gameResults.append(newGameResult)  
  
 # There is still a possibility where only 2 players played the game.  
 # This statement will only add two placements into the database.  
 else:  
 newGameResult = [newGameId, roundCount, winner, runnerUp, "There is no third player in this game."]  
 gameResults.append(newGameResult)  
  
 # Let the players know that the processes finished  
 # and run successfully.  
 print(f"✅ Game Results Added, Game ID: {newGameId}")  
  
 def addWinCount(self, playerId):  
  
 # Increment the win count of a player in the database.  
  
 for player in players:  
 if playerId == player[0]:  
  
 # player[2] = Win Counts  
 player[2] += 1  
 break  
  
def showGameResultsInList():  
  
 # Displays all game results with winner and runner-up details  
 # with a better looking User Interface.  
  
 print("📊 | Game Results:")  
 print("---------------------------------------")  
 print(f"Total Games Played: {len(gameResults)}")  
 print("---------------------------------------")  
  
 # Checks whether if there are data in the database.  
 if len(gameResults) == 0:  
 print("❌ No games have been played yet!")  
 print("Play a game first to see the results.")  
 else:  
 sortedGameResults = sorted(gameResults)  
 for result in sortedGameResults:  
 print(f"Game ID: {result[0]}")  
 print(f"Round(s) played: {result[1]}")  
 print(f"Winner: {result[2]}")  
 print(f"Runner Up: {result[3]}")  
 print(f"Second Runner Up: {result[4]}")  
 print()  
  
 # To let the player able to read and understand all the infos or errors,  
 # An input() function is placed to prevent the code in the next part runs.  
 # The Player can then press <Enter> to proceed on the next part.  
 input("Press <Enter> to continue...")  
  
def showPlayerList():  
  
 # Displays all registered players with their win counts  
 # with a better looking User Interface.  
  
 print("👥 | Player List:")  
 print("-------------------")  
  
 # Checks whether if there are data in the database.  
 if len(players) == 0:  
 print("❌ No players registered yet!")  
 print("Please register players first.")  
 else:  
 for player in players:  
 print(f"Player ID: {player[0]}")  
 print(f"Player Name: {player[1]}")  
 print(f"Winning Count: {player[2]}")  
 print()  
  
 # To let the player able to read and understand all the infos or errors,  
 # An input() function is placed to prevent the code in the next part runs.  
 # The Player can then press <Enter> to proceed on the next part.  
 input("Press <Enter> to continue...")  
  
def gameProcess():  
  
 # Runs the game processes and manage player turns.  
  
 db = DataHandler()  
  
 # Set an immutable value with a fixed name  
 # to prevent type errors and provide clarity.  
 MAX\_SCORE = 100  
  
 # Set a mutable variable to calculate  
 # the total rounds played.  
 roundCount = 1  
 print("🔃 | Processing game validity...")  
 print()  
  
 # Checks if the registered player is more than 2.  
 # If not, the game won't start.  
 if len(players) < 2:  
 print("❌ | This game is not valid to start.")  
 print("❓ | Reason: Player count must be larger than 2.")  
 print()  
  
 # If the players is more than 2,  
 # the game will be allowed to start.  
 else:  
  
 # Convert the player value into a list.  
 # And adds another value to indicate players' position.  
 newPlayers = [ list(player) + [0] for player in players ]  
 print()  
 print("✅ | This game is valid to start.")  
 print("-------------------------------------")  
 print("🐍 | Game Start! | 🪜")  
 print()  
  
 # Make sure it runs indefinitely  
 while True:  
 gameOver = False  
  
 # To iterate in each value in newPlayers.  
 # Which can let every player in newPlayers play the game.  
 for player in newPlayers:  
  
 # Set a name for the attribute of the player  
 # to provide clarity.  
 playerName = player[1]  
  
 # Shows whose turn in the current turn.  
 print(f"🟨 | It's {playerName} turns.")  
  
 # Players can either press <Enter> to roll the dice  
 # or type 'e' to exit.  
 # This can provide more authority to players if they  
 # have emergency issues that require them to end the game.  
 response = input(">> Press <Enter> to roll the dice or 'e' to exit...")  
 if response == "e":  
  
 # The Placement system is still implemented in here  
 # to show the ranks between players.  
 placement = placementIdentifier(newPlayers)  
  
 # This function can handle two different cases  
 # - 2 players  
 # - More than 3 players  
 if len(placement) < 3:  
 db.addGameResults(roundCount, placement[0], placement[1], "")  
 else:  
 db.addGameResults(roundCount, placement[0], placement[1], placement[2])  
  
 print("⛔ | Game Ended")  
 gameOver = True  
 break  
 else:  
 # Set a name for the attribute of the player  
 # to provide clarity.  
 playerId = player[0]  
  
 # Use random.randint() function from 'random' modules  
 # to pick a random number between a given number range.  
 dice = random.randint(1, 6)  
 print()  
  
 # Handles dice roll error  
 if dice < 1 or dice > 6:  
 print(f"❌ Dice Roll Error: Invalid Value -> {dice}")  
 else:  
 print(f"🎲 | {playerName} rolled a {dice}!")  
  
 # Adds the rolled dice value to the  
 # player's current position.  
 player[-1] += dice  
  
 # If the player position is bigger than 100,  
 # It will lessen it to 100.  
 if player[-1] > MAX\_SCORE:  
 player[-1] = MAX\_SCORE  
  
 # This function can handle the Snakes and Ladders function  
 # If the player's current position is matched to the LADDERS' key,  
 # The player's current position will later change to the matched value  
 # from the key.  
 if player[-1] in LADDERS:  
 print(f"🪜 | Yay! {playerName} climbs on a ladder!")  
 player[-1] = LADDERS[player[-1]]  
  
 # Else If the player's current position is matched to the SNAKES' key,  
 # The player's current position will later change to the matched value  
 # from the key.  
 elif player[-1] in SNAKES:  
 print(f"🐍 | Oh no! {playerName} get bitten by a snake!")  
 player[-1] = SNAKES[player[-1]]  
  
 print(f"{playerName} is now at {player[-1]}.")  
 print()  
  
 # If the player's current position is equals to 100  
 # after the processes,  
 # This function will proceed to the winner ceremony session  
 # and record the results and win count of the game.  
 if player[-1] == MAX\_SCORE:  
 gameOver = True  
 placement = placementIdentifier(newPlayers)  
 db.addWinCount(playerId)  
  
 # This function can handle two different cases  
 # - 2 players  
 # - More than 3 players  
 if len(placement) < 3:  
 db.addGameResults(roundCount, placement[0], placement[1], "")  
 else:  
 db.addGameResults(roundCount, placement[0], placement[1], placement[2])  
  
 # Shows rounds played to the players.  
 print(f"👾 | Rounds played: {roundCount}.")  
  
 # Winner ceremony.  
 print(f"🏆 | Congrats to {placement[0]}. {placement[0]} is the winner!")  
 print(f"🥈 | And the runner up goes to ... {placement[1]}!")  
  
 # This code will only be shown if there are  
 # a second runner-up exists.  
 if len(placement) >= 3:  
 print(f"🥉 | Don't forget about our second runner up ... {placement[2]}!")  
  
 print()  
 # Stop the indefinite loop.  
 break  
  
 # When the game is over, the indefinite loop will be canceled.  
 if gameOver:  
 break  
  
 # If the game hasn't over, this function will show the  
 # progress of each player and increment the roundCount  
 # by 1.  
 else:  
 print("📊 | Players Current Progress:")  
 print(f"🎮 | Round(s) Played >> {roundCount}.")  
 roundCount += 1  
 for player in newPlayers:  
 playerName = player[1]  
 playerPosition = player[-1]  
  
 print(f"{playerName} >> {playerPosition}")  
  
 input("Press <Enter> to continue...")  
 print()  
  
 input("Press <Enter> to continue...")  
  
def placementIdentifier(players):  
  
 # Sort players based on their board position  
 # and return ranking order.  
  
 for i in range(len(players)):  
  
 # Assume the first player has the highest rank.  
 highestPoint = i  
  
 # Find the actual highest rank in the remaining  
 # players.  
 for j in range(i + 1, len(players)):  
  
 # Compares if the next player has a larger value  
 # than the current player.  
 if players[j][-1] > players[highestPoint][-1]:  
  
 # Updates the highestPoint if there is another player  
 # that has higher points from the first player.  
 highestPoint = j  
  
 # Swap the actual rank with the current assumed highest rank.  
 players[i], players[highestPoint] = players[highestPoint], players[i]  
  
 # Returns the sorted list based on the ranking.  
 return [player[1] for player in players]  
  
 # Assumes that,  
 # players = [  
 # ['P001', 'Alice', 5, 45], --> Position: 45  
 # ['P002', 'Bob', 3, 89], --> Position: 89  
 # ['P003', 'Charlie', 2, 67] --> Position: 67  
 # ]  
 #  
 # When (i=0),  
 # [Alice(45), Bob(89), Charlie(67)]  
 # ^ ^  
 # i = 0 j = i + 1  
 # j = 1  
 #  
 # - highestPoint starts at 0 (Alice)  
 # - Check if j = 1: Bob(89) > Alice(45)? --> YES --> highestPoint = j = 1  
 # - Check if j = 2: Charlie(67) > Bob(89)? --> NO --> highestPoint = 1  
 #  
 # - Swapping Process:  
 # - players[i], players[highestPoint] = players[highestPoint], players[i]  
 # - players[i] = players[highestPoint] AND players[highestPoint] = players[i]  
 # - players[i] = players[1] players[highestPoint] = players[0]  
 # - players[i] = Bob(89) players[highestPoint] = Alice(45)  
 # - players[0] = Bob(89), players[1] = Alice(45)  
 #  
 # - Result: [Bob(89), Alice(45), Charlie(67)]  
 # - ^  
 # - First  
 # - Now Bob is placed in the first spot of the list.  
 # - After that, the code will then repeat the process several times  
 # - until it is sorted with ranks ascendingly.  
  
def main():  
 while True:  
 print("============================================================================================================================")  
 print("███████╗███╗ ██╗ █████╗ ██╗ ██╗███████╗ █████╗ ███╗ ██╗██████╗ ██╗ █████╗ ██████╗ ██████╗ ███████╗██████╗ ")  
 print("██╔════╝████╗ ██║██╔══██╗██║ ██╔╝██╔════╝ ██╔══██╗████╗ ██║██╔══██╗ ██║ ██╔══██╗██╔══██╗██╔══██╗██╔════╝██╔══██╗")  
 print("███████╗██╔██╗ ██║███████║█████╔╝ █████╗ ███████║██╔██╗ ██║██║ ██║ ██║ ███████║██║ ██║██║ ██║█████╗ ██████╔╝")  
 print("╚════██║██║╚██╗██║██╔══██║██╔═██╗ ██╔══╝ ██╔══██║██║╚██╗██║██║ ██║ ██║ ██╔══██║██║ ██║██║ ██║██╔══╝ ██╔══██╗")  
 print("███████║██║ ╚████║██║ ██║██║ ██╗███████╗ ██║ ██║██║ ╚████║██████╔╝ ███████╗██║ ██║██████╔╝██████╔╝███████╗██║ ██║")  
 print("╚══════╝╚═╝ ╚═══╝╚═╝ ╚═╝╚═╝ ╚═╝╚══════╝ ╚═╝ ╚═╝╚═╝ ╚═══╝╚═════╝ ╚══════╝╚═╝ ╚═╝╚═════╝ ╚═════╝ ╚══════╝╚═╝ ╚═╝")  
 print("============================================================================================================================")  
 print(" at BoardGame Hub, by Jaden & David ")  
 print("============================================================================================================================")  
 print("🏡 MAIN MENU")  
 print("[1] -> ➕ Register new player")  
 print("[2] -> ✏️ Edit player info")  
 print("[3] -> 🗑️ Delete a player")  
 print("[4] -> 👥 Show pass game results")  
 print("[5] -> 🎮 Show all players info")  
 print("--------------------------------")  
 print("[s] -> 👾 Start game")  
 print("[e] -> 🔚 Exit")  
 print("--------------------------------")  
 print("Type your choice (1 - 5 | s | e) to proceed")  
 playerResponse = input(">> ")  
  
 db = DataHandler()  
  
 if playerResponse == "1":  
 db.registerUser()  
 elif playerResponse == "2":  
 db.editPlayersInfo()  
 elif playerResponse == "3":  
 db.deletePlayers()  
 elif playerResponse == "4":  
 showGameResultsInList()  
 elif playerResponse == "5":  
 showPlayerList()  
 elif playerResponse == "s" or playerResponse == "S":  
 gameProcess()  
 elif playerResponse == "e" or playerResponse == "E":  
 print()  
 print("👋 GoodBye!")  
 print()  
 input("Press <Enter> to continue...")  
 break  
 else:  
 print()  
 print("⛔ Invalid Input! Please try again.")  
 print()  
 input("Press <Enter> to continue...")  
  
main()

Outputs and Screenshots:

Main Menu:

A black screen with white text

AI-generated content may be incorrect.Register Player (Error Demo):

A black screen with white text

AI-generated content may be incorrect.

Register Player (Successful Demo):

A computer screen shot of a black screen

AI-generated content may be incorrect.Edit Player Info (Error Demo):

A black screen with white text

AI-generated content may be incorrect.

Edit Player Info (Successful Demo):

A screenshot of a computer program

AI-generated content may be incorrect.

Players Info (Before):

A screen shot of a computer

AI-generated content may be incorrect.

Player Info (After):

A screen shot of a computer

AI-generated content may be incorrect.

Delete Player (Error and Successful Demo):

A screenshot of a computer program

AI-generated content may be incorrect.

Player Info (After Deleted):

A black screen with white text

AI-generated content may be incorrect.

Starts Game (with One Player only):

A screen shot of a computer

AI-generated content may be incorrect.

Start Game (with Two Players) and Game Processes:

A screenshot of a computer program

AI-generated content may be incorrect.

After finishing a game:

A screenshot of a computer game

AI-generated content may be incorrect.

Game Results After Game:

A screenshot of a computer program

AI-generated content may be incorrect.

Players Info After Game:

A screenshot of a computer program

AI-generated content may be incorrect.

Quitting the game:

A black screen with white text

AI-generated content may be incorrect.