תיאור כללי:

נשתמש במבנה נתונים UnionFind בו נתחזק את הקבוצות.

כל קבוצה תכיל מידע על עצמה: עץ דרגות מאוזן של השחקנים (בעלי level גדול מ0), מספר השחקנים בעלי level = 0 ומערך בגודל scale ששומר כמה שחקנים יש בכל score בקבוצה.

בנוסף נתחזק קבוצה מיוחדת שמכילה את כל השחקנים במערכת.

עץ הדרגות, שממומש בעזרת avl, יסדר את השחקנים לפי level ואז לפי id ובנוסף ישמור בכל צומת מערך בגודל scale ובו מספר השחקנים בכל score בתת-עץ.

לבסוף נשמור ב HashTable את כל השחקנים במערכת הממומשת באמצעות dynamic array וchain-hashing.

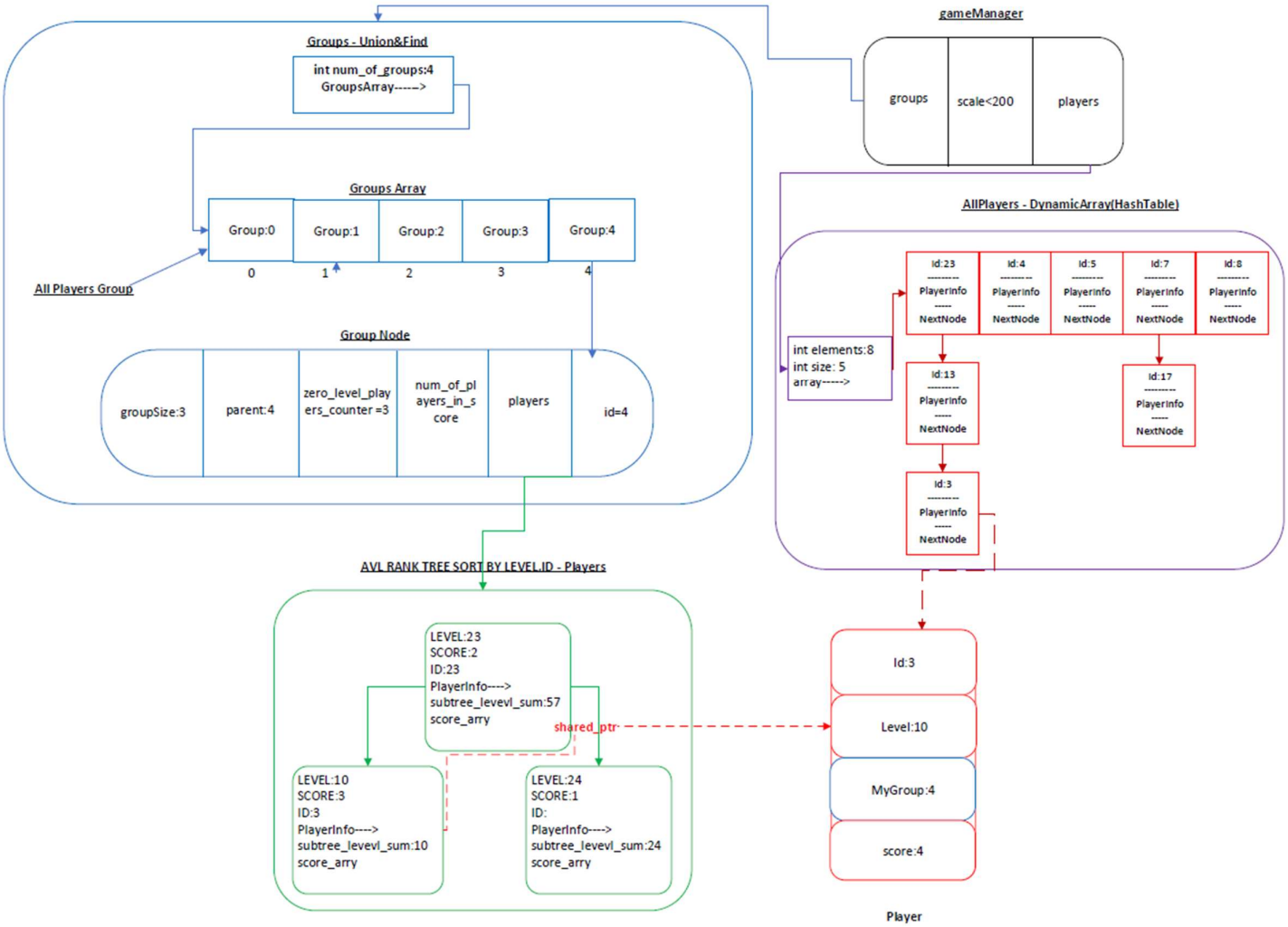
פירוט:

נשמור את המבנה בעצים ובמצביעים הבאים:

1. **GameManager**:
מבנה נתונים המכיל את מערכת המשחק ומנהל אותו:
   a. Players – hash table של שחקני המערכת, המשתמש בchaining עם פונקצית מודולו (כפי שראינו בהרצאות). ה hash table ממומש עם dynamic array.
   b. Scale – integer, מייצג את טווח הניקוד של השחקנים במערכת (מתקבל ביצירת המערכת).
   c. Groups – מבנה מסוג UnionFind שמכיל אובייקט מסוג Group, המייצג את קבוצות השחקנים במערכת.
   המבנה ממומש על ידי מערך של קבוצות, כאשר אינדקס המערך מייצג את id הקבוצה.
   אינדקס 0 שמור לקבוצה מיוחדת ששומרת את כל השחנים במערכת.
2. **Group**:
מחלקה המכילה את פרטי הקבוצה ומידע נוסף עבור הUnionFind:
   a. Size – מספר המייצג את מספר השחקנים בקבוצה.
   b. Parent - מספר המייצג את הid של קבוצת האב של הקבוצה הנוכחית (שווה לid של הקבוצה אם לא התבצע איחוד, אם הקבוצה התמזגה לתוך קבוצה אחרת אז יצביע לid של הקבוצה האחרת).
   c. Zero_level_player_counter – מספר המייצג את מספר השחקנים בקבוצה שהרמה שלהם היא 0.
   d. Num_of_players_in_score – מערך בגודל scale שמראה כמה שחקנים יש בכל score בקבוצה.
   e. Players – עץ דרגות הממומש באמצעות avl tree ומייצגת את השחקנים (בעלי רמה גבוהה מ0) של הקבוצה, לפי סדר של levels ואז id.
   שומר מערך של scores ובו מספר השחקנים בכל score בתת עץ,
   וsubtree_level_sum ששומר את סכום הרמות של השחקנים שנמצאים בתת-עץ.
3. **Players:**
מבנה נתונים המכיל את שחקני המערכת:
   a. Elements – סופר את מספר השחקנים במערכת
   b. Size – שומר את גודל המערך
   c. Array – מבנה מסוג hash table הממומש באמצעות מערך דינאמי ומנגנון chain-hashing המכיל את שחקני המערכת.
4. **Player**:
מבנה נתונים המכיל את פרטי השחקן:
   a. Id – מייצג את הת.ז של השחקן
   b. Level – מייצג את השלב של השחקן

c. Group_id – מייצג את הID של הקבוצה של השחקן
d. Score – מייצג את ניקוד השחקן

ציור


gameManager
groups
scale<200
players
Groups - Union&Find
int num_of_groups:4
GroupsArray----->
Groups Array
Group:0
Group:1
Group:2
Group:3
Group:4
0
1
2
3
4
All Players Group
Group Node
groupSize:3
parent:4
zero_level_players_counter =3
num_of_players_in_score
players
id=4
AllPlayers - DynamicArray(HashTable)
int elements:8
int size: 5
array----->
id:23
PlayerInfo
NextNode
id:4
PlayerInfo
NextNode
id:5
PlayerInfo
NextNode
id:7
PlayerInfo
NextNode
id:8
PlayerInfo
NextNode
id:13
PlayerInfo
NextNode
id:17
PlayerInfo
NextNode
id:3
PlayerInfo
NextNode
AVL RANK TREE SORT BY LEVEL,ID - Players
LEVEL:23
SCORE:2
ID:23
PlayerInfo---->
subtree_levevl_sum:57
score_arry
shared_ptr
LEVEL:10
SCORE:3
ID:3
PlayerInfo---->
subtree_levevl_sum:10
score_arry
LEVEL:24
SCORE:1
ID:
PlayerInfo---->
subtree_levevl_sum:24
score_arry
Id:3
Level:10
MyGroup:4
score:4
Player

פירוט הפונקציות:

1. **Init** – מאתחל מבנה נתונים עם k קבוצות, כל קבוצה מוגדרת להיות ריקה בהתחלה.
   א. אתחול gameManager –
      i. אתחול מערך דינמי Players ריק בגודל קבוע (10)
         - **סיבוכיות:** O(1) (אתחול מערך לגודל קבוע).
      ii. אתחול UnionFind של Groups:
         - אתחול מערך בגודל k: O(k)
         - אתחול קבוצות עם עץ ריק: O(1)
         - **סיבוכיות:** O(k)
      iii. עדכון ערך Scale (חסום ע"י 200) ויצירת מערכים רלוונטים. – O(1).

   סיבוכיות זמן: O(k).
   סיבוכיות מקום: O(k)

2. **MergeGroups** – מאחד 2 קבוצות בUF כפי שנלמד בהרצאה. בנוסף, ממזג את עץ השחקנים לתוך העץ של הקבוצה הגדולה יותר באמצעות merge של עצי avl כפי שנלמד.
   א. שליפת הקבוצות מהמערכת ע"י פונקצית Find (שנלמדה בהרצאה) של UF.
      - **סיבוכיות:** O(log*(k))
   ב. אם הקבוצות כבר מאוחדות, נחזיר "הצלחה", אחרת נמשיך.
   ג. זיהוי הקבוצה הגדול מבין השתיים באמצעות השוואת משתנה size. הגדולה תייצג את הקבוצה המאוחדת
   ד. מיזוג עץ השחקנים מהקבוצה הקטנה לקבוצה הגדולה.
      - **סיבוכיות:** O(n) כאשר n מספר השחקנים בשתי הקבוצות המאוחדות.
   ה. מחיקת עץ השחקנים בקבוצה הקטנה (הקבוצה הגדולה מצביעה על העץ החדש והקטן null).
      - **סיבוכיות:** חסום מלמעלה על ידי O(n)
   ו. עדכון כמות השחקנים בקבוצה המאוחדת החדשה שערך הרמה שלהם הוא 0 וגם את גודל הקבוצה החדשה.
      - **סיבוכיות:** O(1)
   ז. עדכון מערך num_of_players_in_score, בניקוד של השחקנים מהקבוצה הקטנה.
   ח. עדכון parent של הקבוצה הקטנה לparent של הקבוצה הגדולה.

   סיבוכיות זמן: O(log*(k)+n)

   סיבוכיות מקום: O(n) (הקצאת עץ חדש לעץ השחקנים הממוזג, n מספר השחקנים בקבוצות הממוזגות)

3. **AddPlayer** – הכנסת שחקן למערכת המשחק.
   א. ווידוא שהשחקן לא קיים במערכת ע"י פניה לPlayers (hash table) וחיפוש השחקן לפי הid.
      - **סיבוכיות:** O(1) בממוצע על הקלט כפי שראינו בהרצאה
   ב. עדכון הקבוצה אליה מוכנס השחקן -
      i. מציאת הקבוצה באמצעות Find של UF
         - **סיבוכיות:** O(log*(k))
      ii. הגדלת משתנה size של הקבוצה ששומר את מספר השחקנים בקבוצה
      iii. עדכון מערך num_of_players_in_score השומר כמה שחקנים יש מכל score
      iv. הגדלת משתנה zero_level_player_counter השומר על מספר השחקנים ברמה 0
   ג. עדכון הקבוצה עם כל השחקנים - באותו אופן כמו סעיף קודם.
      - **סיבוכיות:** O(1) (אין צורך להשתמש בfind הפעם כדי למצוא את הקבוצה, היא שמורה באינדקס 0 של מערך הUF)
   ד. יצירת שחקן, והכנסתו למערך הדינמי Players
      i. אם כמות השחקנים הקיימת במערכת שווה לגודל המערך – נבצע הרחבה של המערך והעתקת האיברים – ע"י האלגוריתם של DynamicArray שראינו בהרצאה והכנסה ע"י Hashtable.
      ii. הכנסת השחקן החדש למערך הדינאמי ע"י האלגוריתם שראינו בהרצאה (chain-hashing)
      iii. עדכון כמות השחקנים החדשה במערכת (במערך הדינמי).
         - **סיבוכיות:** O(1) משוערך בממוצע על הקלט

סיבוכיות זמן: O(log*(k)) בממוצע על הקלט

סיבוכיות מקום: O(1)

4. **RemovePlayer**– מחיקת שחקן ממערכת המשחק.
   א. ווידוא שהשחקן נמצא במערכת בPlayers (hash table של השחקנים).
      - **סיבוכיות**: O(1) בממוצע על הקלט
   ב. מחיקת שחקן מPlayers
      a. נבצע מחיקה לפי אלגוריתם מחיקה מhash table עם מנגנון chain-hashing
         - **סיבוכיות:** O(1) בממוצע על הקלט
      b. נעדכן את המשתנה השומר את כמות השחקנים בPlayers
      c. אם כמות השחקנים הקיימת במערכת לאחר ההוצאה שווה לרבע מגודל המערך – נבצע כיווץ של המערך והעתקת האיברים – ע"י האלגוריתם של DynamicArray שראינו בהרצאה והכנסה ע"י Hashtable.
         - **סיבוכיות:** O(1) משוערך
   ג. אם רמת השחקן הייתה 0 – נעדכן את כמות השחקנים ברמה 0 בקבוצה שהשחקן היה שייך לה ובקבוצת כל השחקנים
      a. אחרת – ניגשים לקבוצה של השחקן (השמורה בUF) בעזרת פונקציית Find של UF ומוחקים את השחקן מהעץ. באותו אופן ניגש לקבוצה עם id=0 השומרת את כל השחקנים במערכת ונמחק את השחקן מהעץ.
      - **סיבוכיות:** O(log*(k)+log(n)+log(m)) כאשר n – מספר השחקנים בקבוצה וm מספר השחקנים הכוללת במערכת, k מספר הקבוצות
   ד. עדכון גודל הקבוצה וכמות השחקנים ברמת השחקן שהוצא בקבוצת השחקן וקבוצת כל השחקנים (UF Find)

סיבוכיות זמן: O(log*(k)+log(n)) משוערך בממוצע על הקלט כאשר n מספר השחקנים במערכת k מספר הקבוצות

סיבוכיות מקום: O(1)

5. **IncreasePlayerIDLevel** – שינוי level של שחקן.
   א. אם נשלח מצביע לא תקין של המערכת או מזהה שחקן לא חוקי או רמה לא חוקית- נחזיר invalid_input.
   ב. מציאת שחקן במערכת ע"י פניה לPlayers (DynamicArray) וחיפוש השחקן ע"י שימוש באלגוריתם מציאה שנלמד בהרצאה על hash table, אם השחקן לא קיים נחזיר failure.
      - **סיבוכיות**: O(1) בממוצע על הקלט
   ג. מציאת קבוצת השחקן (UF Find)
      - **סיבוכיות:** O(log*(k))
   ד. אם רמת השחקן הקודמת הייתה 0 – הקטנת מספר השחקנים שרמתם 0, בקבוצת השחקן ובקבוצת כל השחקנים (במבנה הקבוצות, שממומש על ידי UF)
   ה. אחרת – הוצאת השחקן מהעצים של הקבוצה שלו וקבוצת כל השחקנים
   ו. עדכון הרמה החדשה על השחקן ועדכון עץ הדרגות (את הscores וסכום הlevels)
   ז. הכנסת השחקן לעצים של הקבוצה שלו ולקבוצת כל השחקנים
      - **סיבוכיות:** O(log(n)+log(m)) כאשר n מספר השחקנים בקבוצה וm מספר השחקנים בכללי

סיבוכיות זמן: O(log*(k)+log(n)) משוערך בממוצע על הקלט כאשר n מספר השחקנים במערכת k מספר הקבוצות

סיבוכיות מקום: O(1)

6. **ChangePlayerIDScore –** שינוי score של שחקן.
   א. אם נשלח מבציע לא תקין של המערכת או מזהה שחקן לא חוקי או ניקוד לא חוקי נחזיר invalid input.
   ב. מציאת שחקן במערכת ע"י פניה לPlayers (DynamicArray) וחיפוש השחקן ע"י שימוש באלגוריתם מציאה שנלמד בהרצאה על hash table - אם לא נמצא השחקן נחזיר failure.

אם נמצא נעדכן את הניקוד.

- **סיבוכיות**: O(1) בממוצע על הקלט

ג. מציאת קבוצת השחקן (UF Find).

- **סיבוכיות:** O(log*(k))

ד. עדכון score_array על קבוצת השחקן וקבוצת כל השחקנים: הקטנת מספר השחקנים שהניקוד שלהם הוא הניקוד הקודם של השחקן, בקבוצת השחקן ובקבוצת כל השחקנים (UF FIND)

ה. אם רמת השחקן שונה מ-0 –

- נמחק את השחקן מהעצים של הקבוצה שלו וקבוצת כל השחקנים
- נעדכן את הניקוד החדש של השחקן
- נוסיף את השחקן לעצים של הקבוצה שלו ולקבוצת כל השחקנים
  - **סיבוכיות:** O(log(n)+log(m)) כאשר n מספר השחקנים בקבוצה וm מספר השחקנים בכללי

ו. הגדלת מספר השחקנים שהניקוד שלהם הוא הניקוד החדש של השחקן, בקבוצת השחקן ובקבוצת כל השחקנים (UF FIND)

סיבוכיות זמן: O(log*(k)+log(n)) משוערך בממוצע על הקלט כאשר n מספר השחקנים במערכת k מספר הקבוצות

סיבוכיות מקום: O(1)

7. **GetPercentOfPlayersWithScoreInBounds** – מחזירה את אחוז השחקנים בטווח levelים נתון עם score נתון, מתוך כלל השחקנים בטווח הlevelים הנתון. בהנתן id של קבוצה שאינו 0 מחזירה עבור הקבוצה, עבור id 0 מחזירה על כל המערכת.

א. עבור קלט לא תקין (קבוצה מחוץ לטווח הקבוצות או פוינטר לא מאותחל) נחזיר invalid input.

ב. נמצא את הקבוצה בUF.

- **סיבוכיות:** O(log*(k))

ג. נמצא את השחקן המינימלי והמקסימלי בטווח הlevelים:

i. נסייר בעץ השחקנים ברקורסיה עד שנמצא את הצומת המתאימה

- **סיבוכיות זמן ומקום**: O(log(n)).

ii. אם level 0 נמצא בטווח נמצא את מספר השחקנים בlevel 0 עם הscore הנתון בכך שנחסיר מהמשתנה העוקב אחרי סך מספר השחקנים עם תוצאה מסוימת (משתנה השמור במערך num_of_players_in_score) את מספר השחקנים עם התוצאה שלא בlevel 0, מידע שנמצא ב root של עץ הדרגות של השחקנים.

- **סיבוכיות:** O(1) כי כל הגישות ישירות (לשורש או למערך במקום נתון)

ד. אם אין שחקנים בטווח נחזיר failure ואחוז 0

ה. אחרת, אם רק שחקנים ברמה 0 קיימים נחזיר את האחוז.

ו. אחרת, נקבל את דרגת השחקנים המקסימלי והמינימלי כדי לחשב כמה שחקנים יש בscore מתחת לשחקן המקסימלי, כמה מתחת למינמלי ולהחסיר ביניהם כדי לקבל בדיוק את הטווח. אם 0 בטווח הlevel-ים נוסיף את משתנה zero_level_players_with_score, מערך ששומר כמה שחקנים ברמה 0 יש מכל תוצאה.

ז. נחשב את האחוז ונחזיר אותו יחד עם success.

סיבוכיות זמן: O(log*(k)+log(n)) משוערך בממוצע על הקלט כאשר n מספר השחקנים במערכת k מספר הקבוצות

סיבוכיות מקום: O(1)

8. **AverageHighestPlayerLevelByGroup** - הפעולה מחשבת את הרמה הממוצעת בה נמצאים m השחקנים ברמות הגבוהות ביותר בקבוצה

א. אם נשלח מצביע לא תקין של המערך או מצביע שמירת הממוצע ו/או מזהה קבוצה לא חוקי ו/או מספר איברים לחישוב לא חוקי- נחזיר invalid input.

ב. מציאת קבוצה במערכת (UF Find) ווידוא שמספר השחקנים לממוצע קטן מגודל הקבוצה

- **סיבוכיות:** O(log*(k))

ג. ניגש לעץ אשר נמצא בקבוצה המיועדת (נזכיר ש0 במבנה נתונים מייצג את קבוצת כל השחקנים) ונשמור את הערך subtree_level_sum של השחקן שנמצא בשורש של העץ – ערך זה מייצג את סכום הרמות של כל השחקנים שנמצאים בעץ (לפי בניית עץ דרגות)

ד. אם כמות השחקנים הנדרשת לחישוב (m) קטנה מכמות השחקנים בעץ ע"י הפונקציה GetSubtreeLevelSumByIndex שמקבלת אינדקס ומחזירה את סכום דרגות כל האיברים בעץ הקטנים מאינדקס בסידור Inorder של העץ (בדומה לאלגוריתם של פונקציית select שראינו בהרצאות).
ניגש לאיבר במקום sizeOfTree-m+1 (האיבר האחרון שנכלל בחישוב) ונשמור את ערכו.
- **סיבוכיות:** O(log(n))

ה. נשמור במצביע הניתן לפונקציה (level) את החישוב הבא-
i. אם נכנסנו לסעיף ד' – אז נחסיר מסכום כל העץ (סעיף ג') את הסכום שקיבלנו בסעיף ד' (כך נקבל את סכום הדרגות של m האיברים האחרונים) ולבסוף נחלק את התוצאה בm
ii. אם לא נכנסו לסעיף ד' – מספיק לחלק את הסכום כל העץ (סעיף ג')ב-m (נשים לב שבמצב הזה אנחנו נדרשים לסכום איברים בקבוצה שערך הרמה שלהם הוא 0 ולכן אין סיבה להוסיפם לסכום)

סיבוכיות זמן: O(log*(k)+log(n)) משוערך בממוצע על הקלט כאשר n מספר השחקנים במערכת k מספר הקבוצות

סיבוכיות מקום: O(1)

9. **Quit** –

א. שחרור Players – hash table הממומש באמצעות מערך דינאמי וchaining– ראינו בהרצאה
- **סיבוכיות:** O(n)

ב. Groups – מעבר קבוצה קבוצה במערך (k) ושחרור העצים כמו האלגוריתם מההרצאה (גודל מקסימלי n)
- **סיבוכיות:** O(k + n)

ג. שחרור המבנה עצמו ומשתנים שהם לא עצים במערכת
- **סיבוכיות:** O(1)

סיבוכיות זמן: O(n+k)

סיבוכיות מקום: O(1)

<u>סיבוכיות מקום של התוכנית:</u> O(n+k) (סכום החישובים של כל פונקציה)