**שמות המגישים:**

עידו אולמר 203428453 [ulmerido@gmail.com](mailto:ulmerido@gmail.com?subject=Domino%20Game:%20ReactJS)

איציק ישר 302732029 [itzikyashar@gmail.com](mailto:itzikyashar@gmail.com)

**ה-URL לניווט מסך הלוגין:**

http://localhost:3000

**החלטות שונות:**

* מהלך – החלטנו שאם שחקן מושך אבן מהקופה, זה נחשב בתור מהלך של השחקן והתור מועבר לשחקן הבא בתור.
* בחרנו להציג גרפית את האפשרות/איסור על מהלך כלשהו על ידי הצגת אבן הדומינו הספציפית בצבעי ירוק/אדום בהתאם.
* בהקשר של התאמת גודל הלוח, על מנת לאפשר חווית שימוש נוחה וידידותית כאשר הלוח נהפך לגדול מידי, החלטנו שבמצבים קיצוניים לאפשר (ללוח בלבד) להיות scrollable. כך הפקדים של המשחק תמיד יהיו באותו מקום ובאותו הגודל ולא יבלבלו את השחקן, המסך לא ישתנה כל הזמן בצורה שמציקה ומפריעה להתמצאות. מה שמאפשר קונססטנטיות באיך שהמשחק נראה וגם הקוביות דומינו נשארות בגודל סביר ונעים לעין.

**הסבר על פונקציות מרכזיות:**

**רקע על הארכיטקטורה של הקוד החדש:**

**צד קליינט:**

הקליינט מורכב מ3 קומפוננטות עיקריות:

**Login** – מרנדרת את מסך הלוגין

**LobbyContainer** – מרנדרת את המסך השני, ומורכבת מקומפוננטות של GameArea שמרנדרת את המשחקים הקיימים בלובי או מ:

**GameContainer** – מרנדרת את מסך המשחק על ידי קומפוננטות של הלוח, אבני משחק, צ'אט, סטטיסטיקות.

**צד שרת:**

השרת מנתב 3 ROUTES:

chatManagement, gameManagement, userManagement.

**chatManagement** – מכיל את בקשות השרת הקשורות לצ'אט

chatManagement.post("/getChat", userAuth.userAuthentication, (req, res) => {

הקליינט מבקש את לוג השיחה על פי שם משחק

chatManagement.route('/postChat')

.post(userAuth.userAuthentication, (req, res) => {

הקליינט מבקש לשלוח הודעה ללוג השיחה על פי שם משחק

chatManagement.route('/restartChat')

.post((req, res) => {

בסוף משחק, אתחול מסך השיחה של המשחק

**userManagement** – מכיל בקשות שרת הקשורות למשתמשים

userManagement.post('/addUser', usersAuth.addUserToAuthList, (req, res) => {

    res.sendStatus(200);

});

בקשה ליצירת משתמש חדש

userManagement.get('/allUsers', usersAuth.userAuthentication, (req, res) => {

res.json(usersAuth.getAllUsers());

});

בקשה לקבלת המשתמשים הקיימים

userManagement.get('/logout', [

    (req, res, next) => {

בקשה למחיקת משתמש קיים

**gameManagement** – מכיל בקשות שרת הקשורות למשחק.

בקשות שקשורות לניהול המשחק עוברות דרך gameAuth

בקשות שקשורות למהלכי המשחק הלוגי עוברות דרך gameLogicAuth

gameManagement.post("/addGame", gamesAuth.addGameToAuthList, (req, res) => {

res.sendStatus(200);

});

בקשה להוספת משחק חדש

gameManagement.post("/removeGame", gamesAuth.removeGameFromAuthList, (req, res) => {

res.sendStatus(200);

});

בקשה למחיקת משחק קיים

gameManagement.post("/joinGame", gamesAuth.addUserToGame, (req,res) => {

res.sendStatus(200);

});

בקשה של משתמש להצטרפות למשחק קיים

gameManagement.post("/watchGame", gamesAuth.addSpectatorToGame, (req,res) => {

res.sendStatus(200);

});

בקשה של צופה לצפות במשחק קיים

gameManagement.post("/leaveGame", gamesAuth.leaveGame, (req,res) => {

res.sendStatus(200);

});

בקשה של משתמש ליציאה ממשחק שנמצא בו

gameManagement.post("/initGame", gamesLogicAuth.initGame, (req,res) => {

res.sendStatus(200);

});

בקשה ראשונית מהקליינט על מנת קבלת מידע ראשוני על משחק חדש

gameManagement.post("/gameStatus", (req,res) => {

בקשה לקבלת מידע על המשחק בכל רגע, נקראת בכל 2 שניות

בנוסף, קיימות בקשות מGAME על מנת לבצע מהלכים במשחק (משיכת קלף, הוספת אבן וכו')

**רקע על הארכיטקטורה של הקוד הלוגי:**הקומפוננטה GAME הנמצאת בקובץ Game.js מכילה ברנדור קומפוננטות אחרות אשר מייצגות אובייקטים כמו board (לוח המשחק) deck (היד של השחקן) Stats (סטטיסטיקות המשחק) ועוד'..

render() { ...

... const myPopup = <Popup

... const myDeck = <Deck

... const myBoard = <Board

הקומפוננטות הפנמיות הנ"ל הן חסרות לוגיקה מורכבת ונועדו רק לשם הצגה בUI. מי שאחראי לעבד ולהביא לקומפוננטה את המידע הנכון אליהן זו הקומפוננטה המכילה אותם (במקרה שלנו, GAME) שמעבירה את המידע הרלוונטי בPROPS.

כל המידע הנחוץ לביצוע מהלכים במשחק והצגת המצב הנוכחי נמצא בstate של קומפוננטת הGAME.

הקופמוננטה מכילה מידע פשוט כמו משתנים בולאנים לגבי האם המשחק התחיל, האם הקופה נגמרה, משספר השחקנים מערך של האבנים בלוח וכו'

בנוסף הקומפוננטה מכילה אובייקטים יותר מורכבים כמן:

this.state = {

history: [], // Array of class History

next: [], [], // Array of class History

player1Stats: new Statistics()};

class Statistics

המחלקה statistics מחזיקה מידע רלוונטי לגבי סטטיסטקות של שחקן במשחק, כמו טיימר (תת מחלקה שמייצגת טיימר של שחקן), ניקוד של השחקן, זמן ממוצע לתור, זמן הכולל מתחילת המשחק וכו'.. למחלקה יש מתודות העוזרות לתפעל את הלוגיקה של הסטיסטיקות של השחקן, כמו להודעי לסטטיסטקה שהתור של השחקן התחיל/הסתיים, עדכון של הנקודות בהתאם ליד של השחקן. השימוש הנכון במחלקה הוא הפעלת הטיימר בתחילת התור

TurnStart()

הפעלת מתודת הUPDATE אחרי מהלך שהשחקן מבצע

Update(playerHand, moveWasDraw)

הפסקת הטיימר בסוף התור

TurnEnd()

למחלקה ממומש גם copy constructor שעוזר לנו בהעתקת המידע לצורף ההיסטוריה.

**על המתודות בGAME:**

למחלקה מספר מתודות handle אשר מטפלות באירועים של הקומפוננטות כמו לחיצה כל כפתורים וכו'...

handleGame(info)

handlePlayerDeck(deck)

handleMouseOut()

handleDrop(e, con)

handleDragOver(e)

handleMouseOver(brick)

handleClickedBrick(brick)

handleDrawClick()

handleUndoClick()

handleNextClick()

handlePrevClick(

handleTimer(info)

dealRandomBricksToPlayer()

מטודה זו מחלקת אקראית מהקופה אבני דומינו לשחקן

isLegalMove(brick)

מטודה זו מקבלת אבן דומינו ומחזירה אמת אם ורק אם אפשר להוסיף את הקוביה הזאת ללוח המשחק.

isLegalDraw()

מתודה זו מחזירה אמת אם ורק אם לשחקן מותר למשוך קוביה מהקופה

nextTurn()

מתודה זו אחראית ללוגיקה של מה קורה בסוף התור (בעתיד תהיה אחראית לניהול התורות במשחק מרובה שחקנים)

updateStatsAfterMove(moveWasDraw)

מתודה שנועדה לעדכן את הסטטיסטקות אחרי שבוצע מהלך כלשהו במשחק

addBrickToBoard(brick)

מוסיפה אבן דומינו ללוח המשחק (ביצוע מהלך של הוספת אבן דומינו)

מתודה עם לוגיקה מורכבת של "טיול" על הלוח ובדיקה האם מותר להניח את אבן הדומינו ובאיזו צורה, ואז גם מבצעת את הפעולה.

saveCurrentStateForHistory

checkEndGame()

בודקת האם המשחק נגמר

**מימוש בונוסים:**

* בונוס 1 – צופה מן הצד.
* בונוס 2 – Chat.