עבודת גמר

# **לקבלת תואר טכנאי תוכנה**

הנושא: **פיתוח משחק Pentago**

#### **המגיש** : רון יצחקי

#### **ת.ז. המגיש**: 328139746

#### **שמות המנחים** : אלון חיימוביץ



אפריל 2025 תשפ"ה

**תקציר:**

המשחק פנטאגו (Pentago) הוא משחק אסטרטגיה בעל מאפיינים ייחודיים המשלבים ראייה מרחבית, תכנון אסטרטגי וחשיבה טקטית המיועד לשני שחקנים שפותח על ידי Tomas Flodén ויצא לאור לראשונה ב-2005 על ידי חברת Mindtwister. המשחק משלב אלמנטים של משחק "חמישה בשורה" (Five in a Row) עם מנגנון ייחודי של סיבוב חלקי הלוח, מה שמוסיף רובד אסטרטגי עמוק.

המשחק מתרחש על לוח בגודל 6 על 6 המחולק לארבעה רביעים של 3×3. ישנם שני שחקנים: שחקן הכלים השחורים (אשר מבצע את המהלך הראשון במשחק) ושחקן הכלים האדומים. בכל תור, השחקן מבצע שני חלקים: ראשית, הנחת כלי משחק במשבצת ריקה כלשהי בלוח, ולאחר מכן סיבוב של אחד מארבעת הרביעים ב-90 מעלות, עם או נגד כיוון השעון. מטרת המשחק היא ליצור רצף של חמישה כלים בצבע שלך בקו ישר (אופקי, אנכי או אלכסוני). המורכבות נובעת מכך שבכל תור, סיבוב הרביע משנה את מבנה הלוח ועשוי לשנות את המצב האסטרטגי בצורה משמעותית. שחקנים נדרשים לחשוב מספר מהלכים קדימה, תוך התחשבות בהשפעות אפשריות של סיבובי הרביעים.

המשחק יכול להסתיים באחת משלוש דרכים:

1. ניצחון לשחקן השחור - כאשר נוצר רצף של חמישה כלים שחורים.
2. ניצחון לשחקן האדום - כאשר נוצר רצף של חמישה כלים אדומים.
3. מצב של תיקו – כאשר הלוח מתמלא לחלוטין מבלי שאף שחקן הצליח ליצור רצף מנצח.

הפרויקט מיישם מערכת הערכת מצבים מתקדמת המסייעת לשחקן המחשב לקבל החלטות אסטרטגיות. מערכת ההערכה מנתחת עמדות משחק באמצעות פרמטרים מגוונים אשר שילובם מעניק ערך מספרי שמייצג את היתרון היחסי של כל שחקן. ערך גדול מעיד על יתרון לשחקן האדום (המחשב), ערך קטן מעיד על יתרון לשחקן השחור (האנושי), וערך קרוב לאפס מצביע על מצב שוויוני.

הפרמטרים להערכת המצב בפרויקט משתייכים לשלושה קטגוריות עיקריות:

1. יצירת איומים ורצפים
2. חסימת איומי היריב
3. השפעת סיבוב הרביעים

**מושגים:**

**משחק אסטרטגיה:**

משחק אסטרטגיה הוא משחק שבו השחקנים, או שחקן יחיד, נדרשים לפעול לפי תוכנית מסודרת במטרה לנצח. תוכנית זה כוללת מטרות ויעדים שהשחקן שואף להשיג במהלך המשחק, יחד עם הדרכים למימושם. בדרך כלל, שחקן עם תוכנית פעולה טובה יותר הוא זה שינצח, קיימים מגוון סוגים של משחקי אסטרטגיה, כגון משחקי אסטרטגיה קבוצתית, אסטרטגיה מופשטת, משחקי מלחמה ועוד. המשחק שבחרתי לעשות עליו את הפרוייקט שלי "Pentago" שייך לקטגוריית משחקי האסטרטגיה המופשטת, משמעות הדבר היא ששני השחקנים מודעים לחלוטין למצב המשחק בכל רגע נתון, הם הגורמים היחידים המשפיעים על התקדמות המשחק, וכל שחקן מבצע בתורו מהלך המורכב משני חלקים קבועים (הנחת כלי וסיבוב רביע). ההצלחה במשחק תלויה אך ורק ביכולת התכנון והתגובה הטקטית של השחקנים.

**הכרעת מצבים:**

במשחקי אסטרטגיה מופשטים כמו פנטגו וכו', מספר המצבים האפשריים במשחק הוא עצום. לכן, לא תמיד אפשר לחפש את כל האפשרוית ולהבין מהו המהלך האופטימלי, ולכן נעשה שימוש בשיטת הכרעת מצבים. בשיטה זו נגדיר עבור המשחק כמה פרמטרים שיעזרו להעריך אם מצבנו טוב יותר או פחות מזה של היריב. באמצעות פרמטרים אלו נבחן את אפשרויות המהלכים שלנו ונבחר את זה שנראה כאופטימלי ביותר. למרות זאת, ייתכן כי המהלך שנבחר לא יהיה בהכרח כהאופטימלי ביותר, שכן שיטה זאת של הכרעת מצבים מבוססת על הערכה מקומית ומוגבלת של ההשלכות העתידיות. אף על פי כן, זו שיטה יעילה וטובה להשגת מהלך שהוא יהיה האופטימלי ביותר.

**בינה מלאכותית והגדרתה:**

בינה מלאכותית (AI) היא תחום במדעי המחשב המתמקד בפיתוח תוכנות ומערכות המסוגלות להפגין התנהגות חכמה – כזו שבעבר נחשבה לייחודית לבני אדם. המחקר בתחום מתמקד בשני כיוונים עיקריים:

1. בינה המדמה חשיבה אנושית, שבה המחשב מחקה את דרך החשיבה והסקת המסקנות של בני אדם. הבוחן המרכזי בתחום זה הוא מבחן טיורינג, שהוצע ב-1950 על ידי אלן טיורינג. מבחן זה בודק האם ניתן להבחין אם משוחח מתקשר עם אדם או עם מכונה דרך ממשק מחשב – ואם לא ניתן להבחין, המכונה תיחשב לחכמה.
2. בינה שאינה מחקה את האדם, אלא מתמקדת ביצירת מנגנוני חשיבה עצמאיים למחשב, ללא קשר לאופן שבו המוח האנושי פועל. סוג זה של AI מתמחה בביצוע משימות מוגדרות בצורה חכמה, כמו במשחק Pentago או פתרון בעיות לוגיות מורכבות.

**ארכיקטורת MVP:**

ארכיטקטורת MVP (Model-View-Presenter) היא תבנית עיצוב אדריכלית המשמשת לפיתוח ממשקי משתמש (UI) בצורה מודולרית, ברורה וניתנת לתחזוקה. לפי ארכיטקטורת MVP, מערכת התוכנה מחולקת למרכיבים נפרדים שכל אחד מהם אחראי על תפקיד מוגדר. החלק שאחראי על הלוגיקה והנתונים הוא ה־Model, והוא משמש כבסיס שמכיל את כל המידע והחוקים של המשחק. ה־View הוא החלק הוויזואלי שאיתו המשתמש נמצא באינטראקציה – הוא אחראי להצגת המידע ולתפיסת קלט מהמשתמש, אך אינו מעבד את המידע בעצמו וה־Presenter משמש כמתווך בין ה־View ל־Model: הוא מקבל מה־View את האירועים שהמשתמש מפעיל, מעבד אותם לפי הצורך, פונה ל־Model כדי לקבל או לשנות מידע, ואז שולח את התוצאה חזרה ל־View להצגה למשתמש.

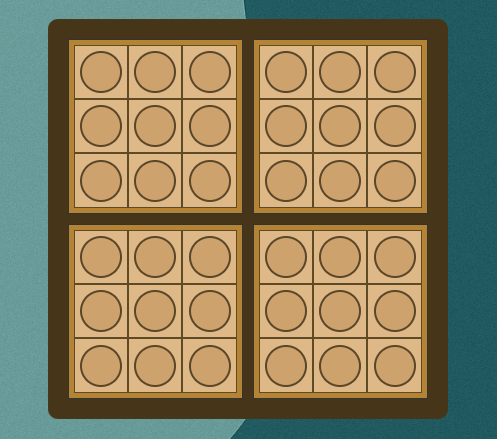
* Model – הרכיב שאחראי על הלוגיקה והחוקים של התוכנה ועל ניהול הנתונים. הוא מעבד מידע, מבצע חישובים, מטפל בבקשות לוגיות, ושומר על הפרדה מלאה מהממשק הגרפי. ה-Model אינו תלוי ב-View או ב-Presenter, דבר המאפשר בדיקות יחידה נפרדות ותחזוקה נוחה יותר.
* View – הרכיב שאחראי על הצגת המידע למשתמש וקבלת קלט ממנו. הוא מציג אך ורק את הממשק הגרפי, מבלי לבצע פעולות לוגיות או החלטות עסקיות. כאשר המשתמש יוזם פעולה (כגון לחיצה על כפתור), ה-View מעביר את האירוע ל-Presenter, ומציג את המידע שהוא מחזיר.
* Presenter – הרכיב המקשר בין ה-View לבין ה-Model. הוא מקבל מה-View את הקלטים של המשתמש, מעבד אותם בהתאם ללוגיקה הנדרשת (למשל, ביצוע בדיקות או טרנספורמציה של נתונים), מתקשר עם ה-Model כדי לקבל או לעדכן מידע, ולבסוף שולח את התוצאה חזרה ל-View להצגה למשתמש.

פיתוח המשחק באמצעות מודל זה של ארכיטקטורת MVP מאפשר הפרדה ברורה בין לוגיקת המשחק, ממשק המשתמש והתיווך ביניהם – מה שתורם לכתיבת קוד נקי, בדיקת יחידות פשוטה, גמישות בשינויים עתידיים וגרימת תכנון מערכות בצורה היעילה ביותר.

**תיאור הנושא:**

**תכולת המשחק:**

המשחק מתרחש על לוח בגודל 6X6, אשר מחולק בצורה ייחודית לארבעה רבעים (לוחות קטנים) נפרדים בגודל 3X3 כל אחד. ישנם שני שחקנים: שחקן הכלים השחורים (המתחיל את המשחק) ושחקן הכלים האדומים (או אדומים, תלוי במימוש היוצר). בניגוד למשחקים כמו שחמט וכו', הכלים (מעין גולות/דיסקיות) אינם מוצבים על הלוח בתחילת המשחק, אלא מתווספים אליו על ידי השחקנים במהלך התור של כל אחד.



**מטרת המשחק:**

מטרת המשחק ב - Pentago היא להיות השחקן הראשון שמצליח ליצור רצף רציף של חמישה כלי משחק בצבע שלו על הלוח. הרצף יכול להיות אופקי, אנכי או אלכסוני, והוא חייב להיווצר כתוצאה ממהלך שלם של השחקן (הנחת כלי ולאחר מכן סיבוב רביע). על מנת להשיג מטרה זו עבור שחקן המחשב (ה-AI), יש צורך באלגוריתם אשר ינתח את המצב הנתון ויבחר את המהלך (צירוף של הנחת כלי וסיבוב רביע) אשר יקדם את השחקן בצורה הטובה ביותר לעבר יצירת רצף מנצח, או לחלופין, ימנע מהיריב להגיע למטרה זו.

**סיום המשחק:**

ישנן שתי אפשרויות לסיום המשחק :Pentago

1. הכרעת המשחק על ידי ניצחון כאשר נוצר רצף של חמישה כלים של אחד השחקנים.
2. הכרעת המשחק על ידי תיקו, מצב זה קורה אם אף אחד מהשחקנים לא הצליח ליצור רצף מנצח של חמישה כלים, והלוח התמלא לחלוטין (כל 36 המשבצות תפוסות). במצב זה, אין עוד מהלכים אפשריים לאף אחד מהשחקנים, והמשחק נגמר ללא מנצח.

**רקע תאורטי:**

למשחק Pentago יש היסטוריה קצרה יחסית אך מעניינת בהתפתחות האלגוריתמים והאסטרטגיות בתחום משחקי הלוח המודרניים. מאז שהוצג לראשונה בשנת 2005 על ידי תומאס פלודן משוודיה, המשחק תפס מקום חשוב בתחום המשחקים האבסטרקטיים התחרותיים, בעיקר בזכות המכניקה הפשוטה שלו והעומק האסטרטגי הנובע מסיבוב הרביעים.

מבחינת פיתוח בינה מלאכותית עבור Pentago, נעשו שימושים שונים באלגוריתמים סטנדרטיים כגון אלגוריתם Minimax ועצי חיפוש מתקדמים, בדומה למה שנעשה במשחקי לוח דומים. בתחילת הדרך, אלגוריתמים אלו שימשו בעיקר למחקר ולניתוח תאורטי של המשחק, אך עם הזמן והתקדמות טכנולוגיית המחשוב בשנות ה-2010 ואילך, הופיעו תוכנות מחשב מתקדמות יותר המסוגלות לשחק Pentago ברמה גבוהה, ואף לנצח שחקנים אנושיים מנוסים.

העניין במשחק גדל במיוחד לאחר שבשנת 2013 הוכח מתמטית שהשחקן הראשון יכול תמיד לכפות ניצחון אם ישחק בצורה מושלמת. עובדה זו הפכה את Pentago לכלי מעניין לחקר אלגוריתמי של אסטרטגיות משחק.

כיום, השימוש בטכניקות של בינה מלאכותית (AI) נפוץ מאוד בפיתוח מערכות המשחק ב-Pentago. מודלים מתקדמים כמו AlphaZero, שהוכיחו הצלחה יוצאת דופן במשחקים אבסטרקטיים, עשויים גם לשמש בסיס למחקר ופיתוח של אלגוריתמי משחק מתוחכמים יותר בעתיד.

פיתוח הבינה המלאכותית למשחק Pentago ממחיש היטב את הקשר בין אלגוריתמים מתקדמים, מחקר תאורטי ופרקטיקה תחרותית, ומדגים כיצד משחק אבסטרקטי פשוט לכאורה יכול להיות כלי רב עוצמה בחקר התחום של AI ומשחקים אסטרטגיים.

**תיאור הבעיה האלגוריתמית:**

ישנן מספר בעיות אלגוריתמיות מרכזיות שיש לפתור בפרויקט זה של המשחק Pentago:

1. יש ליצור ייצוג ממוחשב של משחק הפנטגו לשני שחקנים. הדבר כולל מימוש של אכיפת חוקי המשחק (הנחת כלי במקום פנוי ולאחר מכן סיבוב חובה של אחד הרביעים), וזיהוי מצב הניצחון (יצירת רצף של חמישה כלים) או תיקו (לוח מלא).
2. יש צורך בייצוג יעיל של מצב הלוח. ייצוג זה צריך לאפשר בדיקה מהירה של מהלכים אפשריים (אילו משבצות פנויות, אילו סיבובים חוקיים), גישה נוחה למיקומי הכלים של כל שחקן, ובעיקר – ביצוע יעיל של פעולת סיבוב הרביע ובדיקה מהירה של תנאי הניצחון לאחר כל מהלך.
3. יש ליצור פונקציית הכרעת מצבים. פונקציה זו צריכה לדעת לנתח כל מצב לוח נתון ולהעניק לו ציון מספרי, המבוסס על פרמטרים שונים המעידים על יתרון לאחד השחקנים (כגון פוטנציאל ליצירת רצפים, חסימת רצפי יריב, שליטה אסטרטגית, השפעת סיבובים אפשריים). האלגוריתם של הבינה המלאכותית (AI) צריך להשתמש בפונקציה זו כדי לנתח את התוצאות של כל המהלכים האפשריים בתור הנוכחי ולהמליץ על המהלך (הנחה וסיבוב) המבטיח ביותר. האלגוריתם נדרש לשמור את ההערכה הטובה ביותר שנמצאה עד כה עבור המהלכים שנבדקו, על מנת שיוכל לבחור במהלך הטוב ביותר בסוף הבדיקה.
4. יש להתחשב ביעילות זמן הריצה של פונקציית ההערכה ובחירת המהלך. בכל תור במשחק יש מספר רב של מהלכים אפשריים (צירופים של מיקום הנחה וסיבוב רביע). כדי שה-AI יוכל לבחור מהלך בזמן סביר, פונקציית ההערכה והאלגוריתם שמשתמש בה צריכים להיות יעילים ככל האפשר. יש למצוא דרך לצמצם את כמות המצבים שיש לבדוק, תוך שמירה על איכות בחירת המהלך, כדי שה-AI ימצא מהלך טוב (גם אם לא בהכרח האופטימלי) במהירות.

**סקירת אלגוריתמים בתחום הבעיה:**

ישנה בעיה אלגוריתמיות מרכזית שיש לפתור בפרויקט זה של המשחק Pentago:

* הערכת המצבים: על מנת שה-AI ימצא את המהלך האופטימלי ביותר, פיתחתי מערכת הערכה מבוססת משקולות המאזנת מספר פרמטרים מרכזיים. המערכת מעניקה ציון לכל מהלך אפשרי על פי שורת דירוג המשלבת שליטה במיקומים אסטרטגיים במשחק. חלק מהפרמטרים נותנים יתרון למשחק

מכוון שהם משפרים את עמדתו לנצח בתיאום עם יכולת הסיבוב הייחודית של המשחק,

וחלק מהפרמטרים מציעים חיסרון של המשחק מכיוון שהם מציבים את סיכויי המשחק ל-AI לכישלון.

עם התקדמות המשחק, ערך המשקל נוטה יותר לזיהוי מהלכים מנצחים. ככל שהערך גבוה יותר,

כך לשחקן המחשב יש יתרון גדול יותר. וככל שהערך נמוך יותר, כך לשחקן האנושי יש יתרון גדול יותר, וככל שהערך שואף, לאפס, כך יצויין מצב של תיקו בו לאף אחד מהשחקנים אין יתרון על הצד השני.

**סקירת אלגוריתמים בתחום הבעיה:**

במשחק Pentago קיימים תרחישים רבים אשר ניתן לבצע בכל תור, כאשר לכל תרחיש יש ערך אסטרטגי שונה, בהתאם למצב הלוח ולמהלכים של היריב. כל סיבוב כולל גם הנחת כלי על הלוח וגם סיבוב רבע לוח, מה שהופך את בחירת המהלך למורכבת במיוחד והיכולה להיות היתרון או החיסרון של אחד משני השחקנים המשחקים. עקב כך, הבעיה האלגוריתמית העיקרית בפרויקט המשחק היא ניתוח מצב הלוח הנוכחי ומציאת המהלך האופטימלי ביותר בכל מצב ולאחר מכן מציאת המשבצת ההכרחית אשר בה יש להניח את הכלי של השחקן כך שהוא יגרום ליתרון משמעותי נגד השחקן האויב.

עקב כך נצטרך לעשות שימוש בגישה של הכרעת מצבים - קודם כל נבדק בכל סוף תור של השחקן את מצב הלוח, ובהמשך נשקלות אפשרויות המספקות יתרון אסטרטגי לאורך זמן.

**הכרעת מצבים:**

על מנת לממש את שיטה זו של הכרעת מצבים, נגדיר פרמטרים אסטרטגיים אשר ידרגו כל מצב לוח ובכך נוכל לסרוק את כל המהלכים החוקיים ולבדוק איזה מהם הוא האופטימלי ביותר.

ישנם מספר פרמטרים לדירוג ולהערכת מצב הלוח ביניהם:

1. שליטה במרחב

שליטה במרכז - תפיסת מרכז הלוח מעניקה גמישות ואפשרויות רבות יותר ליצירת רצפים.

שליטה בצמתים אסטרטגיים - מיקומים המחברים בין רביעים שונים מאפשרים יצירת איומים במספר כיוונים.

שליטה ברביעים - איזון בין פיזור כלים במספר רביעים לבין ריכוזם ברביע אחד לצורך שליטה בסיבובים.

2. יצירת איומים מרובים

איומים מקבילים - יצירת מצבים שבהם יש מספר איומים בו-זמנית, מה שמאלץ את היריב להגיב לאחד מהם.

איומים מוסווים - בניית רצפים פוטנציאליים שאינם גלויים מיד, אך יכולים להפוך לאיום לאחר סיבוב.

איומים מתפתחים - בניית רצפים של 2-3 כלים שיכולים להתפתח לאיום ממשי בהמשך.

3. ניצול מכניקת הסיבוב הייחודית

סיבוב התקפי - סיבובים שיוצרים רצף מנצח או מרבים את האיומים שלנו.

סיבוב הגנתי - סיבובים שמפריעים לרצפים של היריב או מונעים איומים.

סיבוב אסטרטגי - סיבובים שמשפרים את המבנה הכולל של הכלים שלנו על הלוח.

4. תזמון ושלבי משחק

פתיחה - התמקדות בבניית בסיס חזק ושליטה בעמדות מפתח.

אמצע משחק - פיתוח דפוסים אסטרטגיים ויצירת איומים ראשוניים.

סיום - זיהוי וניצול מהלכים מנצחים, וחסימת מהלכים מנצחים של היריב.

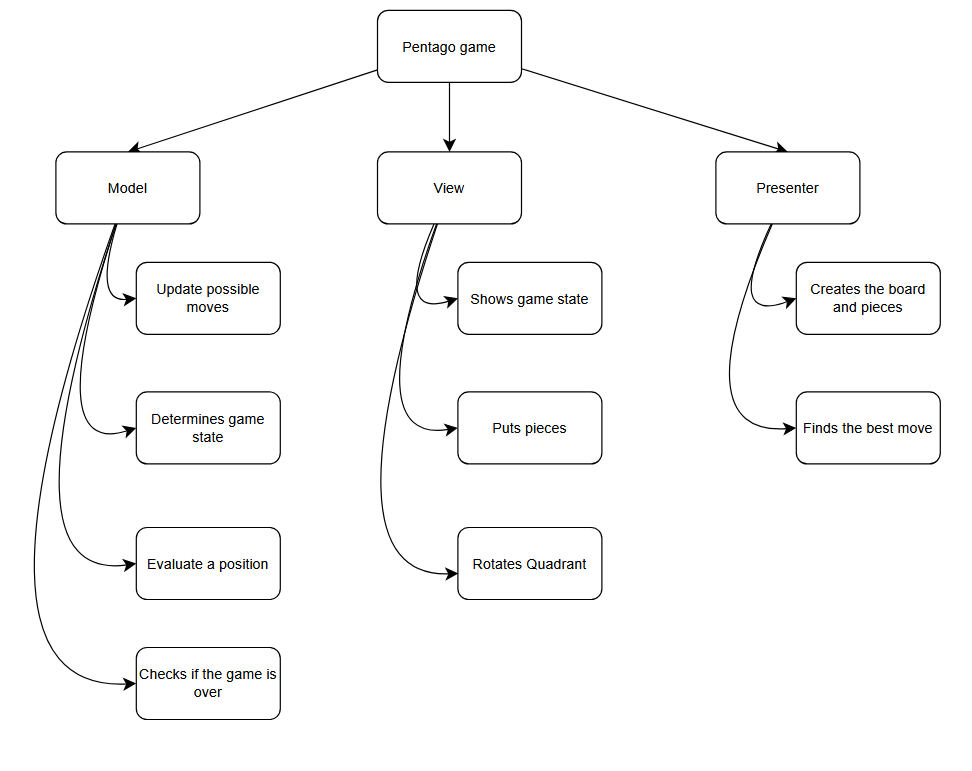
5. איזון בין התקפה להגנה.

עדיפות התקפית - מתי להתמקד ביצירת איומים על חשבון הגנה.

עדיפות הגנתית - מתי להקדיש משאבים לחסימת איומים של היריב.

איזון דינמי - התאמת היחס בין התקפה והגנה בהתאם למצב הלוח.

**ארכיטקטורה של הפתרון המוצע בפורמט Top down level desgin:**



**הסבר על המערכת:**

המערכת של המשחק מורכבת משלושה חלקים אשר מוצגים בתרשים ה – Top Down Level Design שהם למעשה: המודל (Model), התצוגה (View) והמתווך (Presenter).

**המודל (Model):**

1. עדכון המהלכים החוקיים האפשריים במשחק - עדכון המהלכים החוקיים לכל שחקן לאחר הנחת כלי של אותו השחקן על הלוח וסיבוב אחד שלו של אחד מן 4 הרביעים של הלוח.
2. קביעת מצב המשחק - לאחר כל מהלך של שחקן במשחק יש לבדוק את מצב המשחק מפני שיש לו השפעה גדולה ומכריעה על התור הבא של השחקן במידה והמשחק לא הסתיים ויש לו עוד תור.
3. הערכת העמדה ומציאת המהלך האופטימלי ביותר - הערכת העמדה האידיאלית לפי הפרמטרים שהוזכרו לפני וסריקת כל שאר המהלכים החוקיים האפשריים האחרים בכל נקודת זמן במשחק על מנת להעריך את העמדה של השחקן ובכך למצוא את המהלך הטוב ביותר אשר יש לשחק לפיו על מנת לקבל דרך לניצחון.
4. בדיקת סיום המשחק - לאחר כל מהלך מלא של שחקן שהוא הנחת הכלי על הלוח וסיבוב רביע יש לבדוק האם המשחק נגמר בניצחון או הפסד של אחד השחקנים או בתיקו.

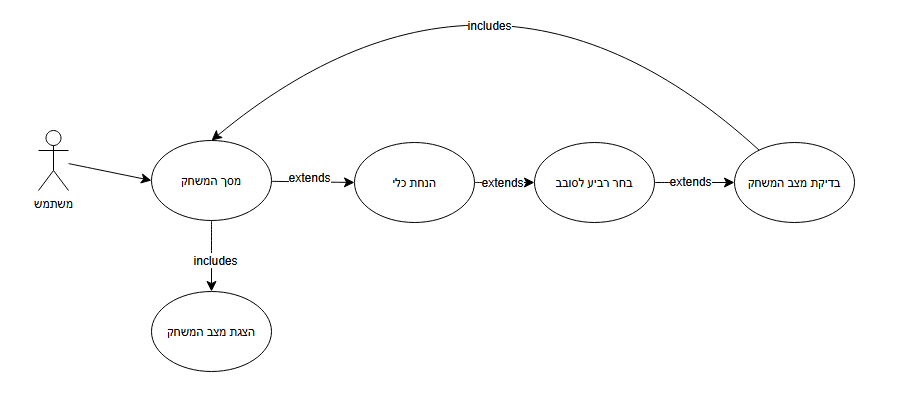
**התצוגה (View):**

1. הצגת מצב המשחק - הצגת 2 מצבי המשחק האפשריים: Winning - מצב ניצחון בו אחד השחקנים הצליח ליצור רצף של 5 כלים מהצבע שלו, Draw - מצב תיקו בו הלוח מתמלא לחלוטין מבלי שאף שחקן הצליח ליצור רצף מנצח של 5 כלים מהצבע שלו.
2. הצבת כלים - מצב בו השחקן מחוייב להניח כלי במשבצת מסויימת, במידה והכלי משוייך לשחקן שהגיע תורו לשחק.
3. סיבוב רביע - מצב בו השחקן מחוייב לאחר הגעת התור אליו והנחת הכלי שלו במשבצת מסויימת לבחור איזה רביע מבין 4 הרביעים שבלוח לסובב עם או נגד כיוון השעון.

**המתווך (Presenter):**

1. יצירת הלוח והכלים - בתחילת המשחק נוצר עצם של לוח אשר עליו צריך להניח את הכלים של כל שחקן ועליו מתקיים מהלך המשחק, הלוח בגודל 6X6 כך שישנן 36 משבצות ריקות לתחילת המשחק והוא מחולק ל - 4 תתי לוח אשר כל אחד בגודל 3X3 על מנת פעולת הסיבוב שנבצע עליהן לאחר הנחת הכלי של השחקן שהגיע תורו. במהלך המשחק נוצרים גם עצמים של הכלים של כל אחד מהשחקנים.
2. מציאת המהלך האופטימלי ביותר - מציאת המהלך הטוב ביותר על ידי ה - AI על ידי שימוש באלגוריתם של הכרעת המצבים FSM בהתבסס על מצב הלוח הנוכחי.

**תרשים מקרי שימוש UML Use Cases:**



**מבנה נתונים:**

**לוח ביטים:**

אלגוריתם זה מכיל משתנים מסוג long המשמשים את המיקומים בלוח. הבינה המלאכותית במשחק שלי משתמשת בייצוג לוח ביטים (Bitboard) כדי לייצג את מצב המשחק באופן יעיל ביותר. בשיטה זו, המצב של כל הלוח 6X6 מיוצג בעזרת שני מספרים ארוכים מסוג long (64 ביט) - אחד עבור כלי השחקן השחור (blackBoard) ואחד עבור כלי השחקן האדום (whiteBoard). באמצעות לוח הביטים אנו יכולים להציג עצמים פיזיים עם מיקומים על הלוח. כאשר ביט דולק (מוגדר כ-1) באחד מהלוחות, הדבר מציין שיש כלי משחק של השחקן המתאים במיקום זה. לדוגמה: אם הביט במיקום ה-17 ב-blackBoard הוא 1, יש כלי שחור בשורה 2, עמודה 5 אם הביט במיקום ה-23 ב-whiteBoard הוא 1, יש כלי לבן/אדום בשורה 3, עמודה 5. היתרון הגדול בייצוג זה הוא שניתן לבצע בדיקות מורכבות (כמו בדיקת ניצחון) באמצעות פעולות ביטיות ביעילות של O(1) במקום לבדוק כל תא בנפרד.

**רשימת לאחסון מהלכים ועמדות:**

השתמשתי ברשימות על מנת להחזיק את כל המהלכים האפשריים במשחק אשר מתעדכנים לאחר כל הנחת כלי וסיבוב רביע. יש חשיבות להחזקת רשימת המהלכים האפשריים עבור הבינה המלאכותית על מנת להעריך את כל האפשרויות ולבחור במהלך האופטימלי, במיוחד כשמדובר בדפוסים אסטרטגיים כמו אלכסונים, דפוס Z ודפוס טבעת. בנוסף, פונקציית הערכת המצבים משתמשת בניתוח של מהלכים פוטנציאליים ליצירת רצפים של חמישה כלים או לחסימת רצפים של היריב, תוך התחשבות באפשרויות הסיבוב השונות של הרביעים. יתר על כן, יש צורך בהערכת המצבים לעבור על כל המהלכים האפשריים ולסמלץ את השפעת סיבובי הרביעים השונים על מנת לבדוק האם העמדה העתידית לאחר הנחת הכלי והסיבוב תהיה אופטימלית יותר מבחינת שליטה במרכז, יצירת איומים והתקדמות לניצחון.

**רשימת מהלכים ועמדות:**

השתמשתי ברשימות על מנת להחזיק את כל המהלכים האפשריים במשחק אשר מתעדכנים לאחר כל הנחת כלי וסיבוב רביע. יש חשיבות להחזקת רשימת המהלכים האפשריים עבור הבינה המלאכותית על מנת להעריך את כל האפשרויות ולבחור במהלך האופטימלי. בנוסף, פונקציית הערכת המצבים משתמשת בניתוח של מהלכים פוטנציאליים ליצירת רצפים של חמישה כלים או לחסימת רצפים של היריב, תוך התחשבות באפשרויות הסיבוב השונות של הרביעים. יתר על כן, יש צורך בהערכת המצבים לעבור על כל המהלכים האפשריים ולסמלץ את השפעת סיבובי הרביעים השונים על מנת לבדוק האם העמדה העתידית לאחר הנחת הכלי והסיבוב תהיה אופטימלית יותר מבחינת שליטה במרכז, יצירת איומים והתקדמות לניצחון.

**מטריצת משקלים:**

מטריצת המשקלים מהווה מרכיב מרכזי במערכת קבלת ההחלטות של הבינה המלאכותית, ומשמשת להערכת הערך האסטרטגי של כל עמדה על לוח המשחק. המטריצה מממשת רעיון מרכזי בתורת המשחקים - לא כל המשבצות שוות בערכן. באמצעות מערך דו-ממדי בגודל 6×6, המקביל למבנה הלוח, מוקצה לכל משבצת ערך מספרי המשקף את חשיבותה האסטרטגית. ערכים גבוהים יותר (8) ניתנים למשבצות מרכזיות המאפשרות שליטה מירבית, ערכים בינוניים (6-4) למשבצות בטבעות הפנימיות והחיצוניות, וערכים נמוכים יותר (2) לאזורי השוליים. בנוסף, ישנה הגדלה ממוקדת של ערכים למשבצות מיוחדות כמו מרכזי הרביעים (+2) ונקודות חיבור אסטרטגיות בין רביעים (+1). בעת בחירת מהלך, המערכת מנתחת את המשבצות הפנויות ובוחרת אלו בעלות המשקל הגבוה ביותר, תוך התחשבות גם במצב המשחק הנוכחי ובמיקומי הכלים האחרים על הלוח.

**מפת דפוסים:**

מפת דפוסים היא למעשה מבנה הנתונים של דפוסים אסטרטגיים המיישם גישה מתקדמת יותר לקבלת החלטות בבינה המלאכותית, המבוססת על זיהוי וניצול תבניות משחק שלמות במקום משבצות בודדות. באמצעות מפה (HashMap) המקשרת בין שמות דפוסים ("diagonal1", "diagonal2", "z\_pattern", "ring\_pattern") לרשימות של קואורדינטות, המערכת מגדירה תצוגות מנצחות או אסטרטגיות שהבינה המלאכותית שואפת לבנות במהלך המשחק. כל דפוס מורכב מסדרה של מיקומים המהווים יחד מבנה אסטרטגי שלם - כמו אלכסון מלא לאורך הלוח, תבנית Z אלכסונית, או טבעת המקיפה את מרכז הלוח. הבינה המלאכותית מנסה לתפוס משבצות מתוך דפוסים אלה כחלק מאסטרטגיה ארוכת טווח, במקום להסתמך רק על ערך המשבצת הבודדת. בשלבים מתקדמים של המשחק, המערכת מזהה באיזו מידה הצליחה להשלים דפוס מסוים, ובהתאם לכך מתעדפת מהלכים שימשיכו לבנות או להשלים את הדפוס. יתרה מכך, בעת שיקולי סיבוב הרביעים, המערכת בוחנת כיצד כל סיבוב אפשרי ישפיע על הדפוסים האסטרטגיים שבונה או מנסה לחסום.

**שפת תכנות וסביבת עבודה:**

שפת תכנות : Java.

סביבת עבודה: IntelliJ IDEA 2024.3 (Community Edition).

**אלגוריתם ראשי (פסיאודו קוד):**

1. אתחול הבינה המלאכותית PentagoAI()

1.1. הגדר מצב התחלתי = CONTROL\_CENTER

1.2. אתחל מונה תורים = 0

1.3. - initializePositionWeights()אתחל מטריצת משקלים לעמדות על הלוח

1.3.1. הגדר משקלים גבוהים למרכז הלוח (8)

1.3.2. הגדר משקלים בינוניים לטבעת הפנימית (6)

1.3.3. הגדר משקלים נמוכים יותר לטבעת החיצונית (4)

1.3.4. הגדר משקלים נמוכים לקצוות ולפינות (2)

1.3.5. הוסף משקל נוסף למרכזי הרביעים

1.3.6. הוסף משקל נוסף לחיבורים בין רביעים

1.4. - initializeStrategicPatterns()אתחל דפוסים אסטרטגיים

1.4.1. אתחל דפוס אלכסוני

1.4.2. אתחל דפוס אלכסון נגדי

1.4.3. אתחל דפוס Z

1.4.4. אתחל דפוס טבעת

2. חישוב מהלך - makeMove()

2.1. הגדל מונה תורים ב-1

2.2. קבע את מצב הבינה המלאכותית - determineState()

2.2.1. אם יש אפשרות לניצחון מיידי, עבור למצב OFFENSE

2.2.2. אם היריב קרוב לניצחון, עבור למצב DEFENSE

2.2.3. אם כדאי לחשב מהלכים קדימה, עבור למצב LOOK\_AHEAD

2.2.4. אם בתחילת המשחק ויש הזדמנות לבנות דפוס, עבור למצב BUILD\_PATTERN

2.2.5. אם יש עמדות אסטרטגיות במרכז, עבור למצב CONTROL\_CENTER

2.2.6. אם יש איום על פינות חשובות, עבור למצב BLOCK\_CORNERS

2.2.7. אם יש הזדמנות לבנות דפוס אסטרטגי, עבור למצב BUILD\_PATTERN

2.2.8. אם כדאי להתמקד בקצוות הלוח, עבור למצב CONTROL\_EDGES

2.2.9. אם בשלב מתקדם במשחק, עבור למצב CONTROL\_ROTATION

2.2.10. אחרת, בחר באקראי

2.3. החלט אם להשתמש בחישוב מהלכים קדימה (הסתברות 80%)

2.4. אם יש צורך בחישוב מהלכים קדימה ובמצב LOOK\_AHEAD

2.4.1. החזר את המהלך הטוב ביותר עם חישוב קדימה - calculateBestMoveWithLookAhead()

a. מצא את כל המהלכים האפשריים

b. עבור כל מהלך אפשרי, חשב ציון המהלך

i. הוסף ציון בסיסי לפי משקל העמדה

ii. בצע מהלך בלוח זמני

iii. בדוק איומים שנוצרו ואיומים שנחסמו

iv. בדוק אפשרויות סיבוב שיובילו לניצחון

v. הוסף בונוס אם המהלך הוא חלק מדפוס אסטרטגי

c. החזר את המהלך עם הציון הגבוה ביותר

2.5. אחרת, בחר מהלך לפי המצב הנוכחי

2.5.1. במצב OFFENSE - חפש מהלך התקפי

a. חפש מהלכים שמובילים לרצף של 4

b. אם אין, חפש מהלכים שמובילים לרצף של 3

c. אם אין, חזור למהלך אסטרטגי

2.5.2. במצב DEFENSE - חפש מהלך הגנתי

a. חפש מהלכים שחוסמים רצף של 4 של היריב

b. אם אין, חפש מהלכים שחוסמים רצף של 3 של היריב

c. אם אין, חזור למהלך אסטרטגי

2.5.3. במצב CONTROL\_CENTER - חפש מהלך אסטרטגי במרכז

a. נסה לתפוס עמדות במרכז הלוח (2x2)

b. אם אין, נסה לתפוס מרכזי רביעים

c. אם אין, חזור למהלך בקצוות

2.5.4. במצב CONTROL\_EDGES - חפש מהלך בקצוות

a. נסה לתפוס עמדות בשורות ועמודות קצה

b. אם אין, נסה לתפוס פינות

c. אם אין, חזור למהלך אקראי

2.5.5. במצב BLOCK\_CORNERS - חפש מהלך לחסימת פינות

a. אם היריב תפס פינה, נסה לחסום התקדמות ממנה

b. אם אין, נסה לחסום גישה לפינות

c. אם אין, חזור למהלך אסטרטגי

2.5.6. במצב BUILD\_PATTERN - חפש מהלך דפוס אסטרטגי

a. בחר את הדפוס האסטרטגי המתאים ביותר למצב המשחק

b. חפש עמדות פנויות בדפוס הנבחר

c. תעדף עמדות שמרחיבות דפוסים קיימים

2.5.7. במצב CONTROL\_ROTATION - חפש מהלך המתכונן לסיבוב

a. זהה רביעים עם הכי הרבה כלים

b. נסה לשים כלי במשבצת פנויה ברביע נבחר

c. אם אין, חזור למהלך אסטרטגי

2.5.8. אחרת - בחר מהלך אקראי מבין המהלכים החוקיים

3. חישוב סיבוב - makeRotation()

3.1. קבע אם להשתמש בניתוח מעמיק (הסתברות 80%)

3.2. אם משתמשים בניתוח מעמיק

3.2.1. נסה למצוא סיבוב התקפי - findOffensiveRotation()

a. בדוק את כל הרביעים והכיוונים

b. סמלץ כל סיבוב על לוח זמני

c. בדוק אם הסיבוב יוצר ניצחון

d. אם כן, החזר את הסיבוב הזה

3.2.2. אם לא נמצא סיבוב התקפי, נסה למצוא סיבוב הגנתי

a. בדוק את כל הרביעים והכיוונים

b. סמלץ כל סיבוב על לוח זמני

c. בדוק אם הסיבוב מונע ניצחון של היריב

d. אם כן, החזר את הסיבוב הזה

3.2.3. אם לא נמצא סיבוב הגנתי, נסה למצוא סיבוב אסטרטגי משופר

a. בדוק את כל הרביעים והכיוונים

b. סמלץ כל סיבוב על לוח זמני

c. הערך את מצב הלוח לאחר הסיבוב

i. בדוק השפעה על איומים שלנו

ii. בדוק השפעה על איומי היריב

iii. בדוק השפעה על דפוסים אסטרטגיים

iv. בדוק השפעה על עמדות אסטרטגיות

d. החזר את הסיבוב עם הציון הגבוה ביותר

3.2.4. אם לא נמצא סיבוב אסטרטגי, מצא סיבוב חכם אקראי

a. תעדף רביעים עם יותר כלים של היריב

b. בחר כיוון סיבוב אקראי

3.3. אחרת

3.3.1. נסה למצוא סיבוב התקפי

3.3.2. אם לא נמצא, נסה למצוא סיבוב הגנתי

3.3.3. אם לא נמצא, בחר רביע וכיוון אקראיים

4. החזר את הסיבוב הנבחר והעבר את התור לשחקן האנושי

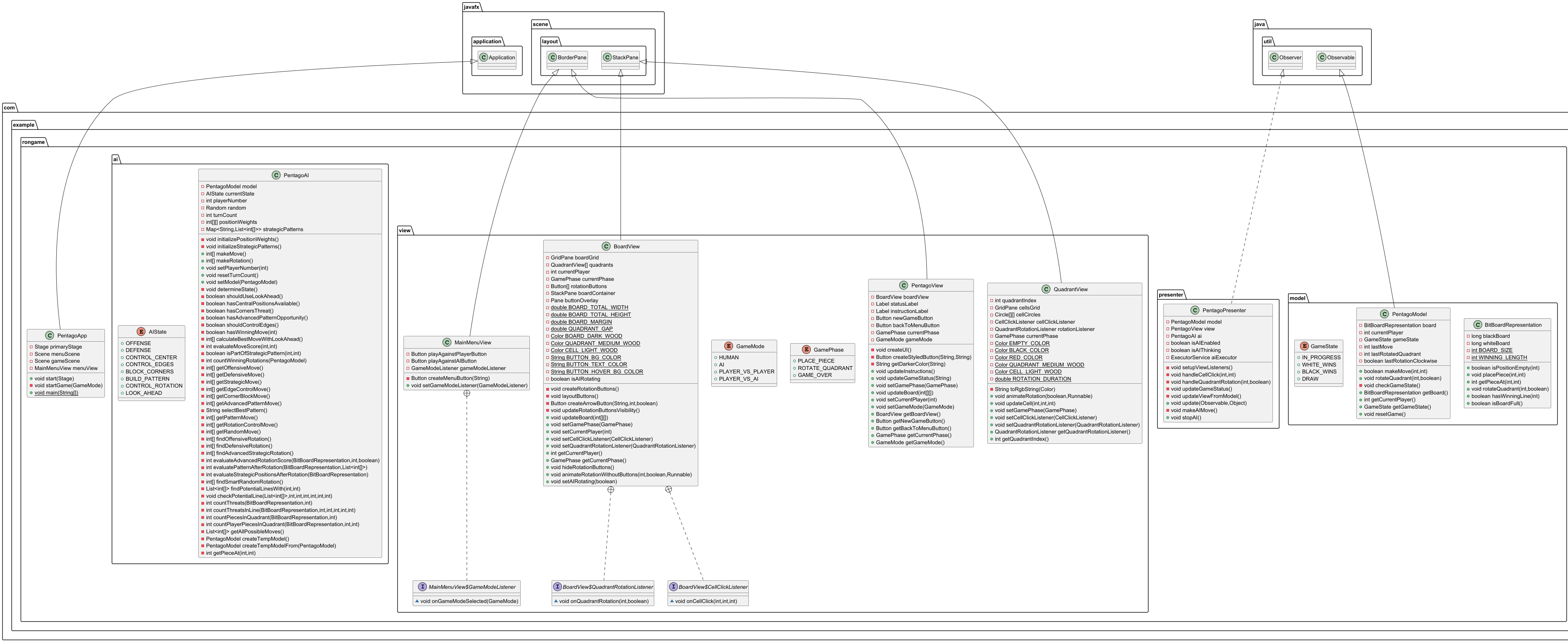
**תיאור ממשק גרפי:**

בפרויקט השתמשתי ב - SCENE BUILDER על מנת לייצר את הגרפיקה (GUI) של המשחק, מה שאפשר לי לעצב את הממשק באופן ויזואלי וליצור ממשק משתמש גרפי אינטראקטיבי. תוכנת המשחק Pentago מורכבת ממסך משחק ראשי המציג לוח 6X6 המחולק לארבעה רביעים של 3X3. כל משבצת בלוח היא רכיב גרפי שמסוגל להציג את מצבה (ריקה, כלי שחור, או כלי אדום).

רכיבים מרכזיים במשחק כוללים את לוח המשחק, כפתורי הסיבוב לכל רביע, ותצוגת סטטוס המשחק. כל רכיב במשחק יורש ממחלקת בסיס המספקת פונקציונליות לרכיבים, טיפול באירועים, ועדכון המצב הגרפי. הממשק מאפשר אינטראקציה אינטואיטיבית עם המשתמש לביצוע פעולות המשחק. השתמשתי ברקעים מיוחדים שיצרתי בפוטושופ להעשרת החוויה הויזואלית ולהדגשת אלמנטים חשובים במשחק.

השחקן מבצע מהלך על ידי לחיצה על משבצת ריקה להנחת כלי חדש, ולאחר מכן בוחר באחד מהרביעים וכיוון סיבוב (עם או נגד כיוון השעון). הממשק הגרפי מתעדכן בזמן אמת להצגת המהלכים שבוצעו, תוך שימוש באנימציות תזוזה שמדגישות את סיבוב הרביע הנבחר לאחר הנחת הכלי, ומעביר את התור לשחקן הבא.

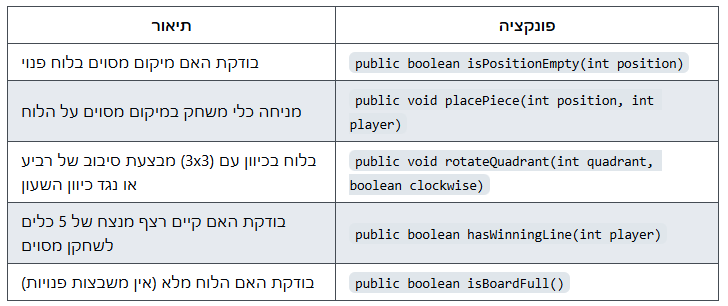
**UML Class Diagram:**



**תיאור המחלקות הראשיות והפונקציות הראשיות בפרויקט:**

**שם מחלקה: BitBoardRepresentation**

**תיאור מחלקה:** המחלקה מיישמת ייצוג יעיל של לוח משחק 6X6 באמצעות ביטים (bitboard). המחלקה משתמשת בשני משתנים מסוג long (64 ביט), blackBoard ו-whiteBoard, כדי לייצג את מיקום כלי המשחק של שני השחקנים. כל ביט במספר מייצג משבצת ספציפית בלוח, כאשר ערך 1 מציין נוכחות של כלי משחק ו-0 מציין משבצת ריקה. המחלקה מספקת פונקציות לניהול המשחק כמו הנחת כלים, סיבוב רביעים של הלוח, ובדיקת מצבי ניצחון.



**public boolean isPositionEmpty(int position):**

**הפרמטרים:** מיקום על הלוח.

**מה הפונקציה מחזירה:** מחזירה true אם המיקום פנוי, ו-false אם המיקום תפוס.

**מה הפונקציה מבצעת:** בודקת האם משבצת מסוימת בלוח פנויה.

**יעילות הפונקציה:** (1)O - פעולת ביט אחת בלבד.

**תיאור פסיאודו קוד של הפונקציה:**

1. יצירת מסכה בינארית עם ביט דלוק במיקום המבוקש

1.1. הזזת הביט 1 למיקום הרצוי

2. בדיקה האם הביט במיקום זה דלוק באחד מלוחות השחקנים

2.1. חישוב איחוד ביטים של שני הלוחות

2.2. ביצוע פעולת AND בין התוצאה לבין המסכה

2.3. בדיקה האם התוצאה היא 0

3. החזרת תוצאת הבדיקה

**public void placePiece(int position, int player):**

**הפרמטרים:** מיקום על הלוח וזיהוי השחקן (0 לשחקן שחור, 1 לשחקן אדום).

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מניחה כלי משחק במיקום מסוים עבור השחקן המבוקש.

**יעילות הפונקציה:** (1)O - פעולת ביט אחת בלבד.

**תיאור פסיאודו קוד של הפונקציה:**

1. יצירת מסכה בינארית עם ביט דלוק במיקום המבוקש

1.1. הזזת הביט 1 למיקום הרצוי

2. הוספת הכלי ללוח השחקן המתאים

2.1. אם מדובר בשחקן הראשון

2.1.1. הדלקת הביט המתאים בלוח השחקן הראשון

2.2. אחרת (השחקן השני)

2.2.1. הדלקת הביט המתאים בלוח השחקן השני

**public void rotateQuadrant(int quadrant, boolean clockwise):**

**הפרמטרים:** מספר הרביע לסיבוב וכיוון הסיבוב (true לכיוון השעון, false נגד כיוון השעון).

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מסובבת רביע של 3X3 משבצות בלוח בכיוון הנדרש.

**יעילות הפונקציה:** (1)O - מספר קבוע של פעולות.

**תיאור פסיאודו קוד של הפונקציה:**

1. חישוב נקודת ההתחלה של הרביע

1.1. חישוב שורת ההתחלה לפי מספר הרביע

1.2. חישוב עמודת ההתחלה לפי מספר הרביע

2. שמירת המצב הנוכחי של הרביע במערך זמני

2.1. יצירת מערך דו-ממדי 3x3

2.2. עבור כל שורה ברביע

2.2.1. עבור כל עמודה ברביע

2.2.1.1. שמירת ערך המשבצת במערך הזמני

3. הסרת הכלים הקיימים ברביע

3.1. עבור כל שורה ברביע

3.1.1. עבור כל עמודה ברביע

3.1.1.1. חישוב המיקום המלא בלוח

3.1.1.2. יצירת מסכה להסרת הביט

3.1.1.3. הסרת הביט מלוחות שני השחקנים

4. הנחת הכלים בסיבוב

4.1. עבור כל שורה במערך הזמני

4.1.1. עבור כל עמודה במערך הזמני

4.1.1.1. חישוב המיקום החדש לפי כיוון הסיבוב

4.1.1.1.1. אם הסיבוב בכיוון השעון

4.1.1.1.1.1. שורה חדשה = העמודה המקורית

4.1.1.1.1.2. עמודה חדשה = גודל רביע - שורה מקורית - 1

4.1.1.1.2. אחרת (סיבוב נגד כיוון השעון)

4.1.1.1.2.1. שורה חדשה = גודל רביע - העמודה המקורית - 1

4.1.1.1.2.2. עמודה חדשה = השורה המקורית

4.1.1.2. אם המשבצת לא ריקה

4.1.1.2.1. חישוב המיקום החדש בלוח המלא

4.1.1.2.2. הנחת הכלי במיקום החדש

**public boolean hasWinningLine(int player):**

**הפרמטרים:** זיהוי השחקן (0 לשחקן שחור, 1 לשחקן אדום).

**מה הפונקציה מחזירה:** מחזירה true אם השחקן יצר רצף מנצח, ו-false אחרת.

**מה הפונקציה מבצעת:** בודקת אם יש רצף של 5 כלים משחק בשורה, עמודה או אלכסון לשחקן מסוים.

**יעילות הפונקציה:** (1)O - מספר קבוע של פעולות.

**תיאור פסיאודו קוד של הפונקציה:**

1. בחירת הלוח המתאים לשחקן

1.1. אם מדובר בשחקן הראשון, בחירת הלוח הראשון, אחרת בחירת הלוח השני

2. בדיקת רצפים בשורות

2.1. עבור כל שורה בלוח

2.1.1. עבור כל עמודת התחלה אפשרית

2.1.1.1. איפוס משתנה הצלחה

2.1.1.2. עבור כל תא ברצף הנדרש

2.1.1.2.1. חישוב מיקום התא בלוח

2.1.1.2.2. בדיקה האם הביט במיקום זה דלוק

2.1.1.2.3. אם לא, הורדת דגל ההצלחה ויציאה מהלולאה הפנימית

2.1.1.3. אם כל הביטים ברצף דלוקים, החזרת אמת

3. בדיקת רצפים בעמודות

3.1. עבור כל עמודה בלוח

3.1.1. עבור כל שורת התחלה אפשרית

3.1.1.1. איפוס משתנה הצלחה

3.1.1.2. עבור כל תא ברצף הנדרש

3.1.1.2.1. חישוב מיקום התא בלוח

3.1.1.2.2. בדיקה האם הביט במיקום זה דלוק

3.1.1.2.3. אם לא, הורדת דגל ההצלחה ויציאה מהלולאה הפנימית

3.1.1.3. אם כל הביטים ברצף דלוקים, החזרת אמת

4. בדיקת רצפים באלכסונים (מימין לשמאל)

4.1. עבור כל שורת התחלה אפשרית

4.1.1. עבור כל עמודת התחלה אפשרית

4.1.1.1. איפוס משתנה הצלחה

4.1.1.2. עבור כל תא ברצף הנדרש

4.1.1.2.1. חישוב מיקום התא באלכסון

4.1.1.2.2. בדיקה האם הביט במיקום זה דלוק

4.1.1.2.3. אם לא, הורדת דגל ההצלחה ויציאה מהלולאה הפנימית

4.1.1.3. אם כל הביטים ברצף דלוקים, החזרת אמת

5. בדיקת רצפים באלכסונים (משמאל לימין)

5.1. עבור כל שורת התחלה אפשרית

5.1.1. עבור כל עמודת התחלה אפשרית

5.1.1.1. איפוס משתנה הצלחה

5.1.1.2. עבור כל תא ברצף הנדרש

5.1.1.2.1. חישוב מיקום התא באלכסון ההפוך

5.1.1.2.2. בדיקה האם הביט במיקום זה דלוק

5.1.1.2.3. אם לא, הורדת דגל ההצלחה ויציאה מהלולאה הפנימית

5.1.1.3. אם כל הביטים ברצף דלוקים, החזרת אמת

6. אם לא נמצא רצף מנצח, החזרת שקר

**public boolean isBoardFull():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** מחזירה true אם הלוח מלא, ו-false אם יש משבצות פנויות.

**מה הפונקציה מבצעת:** בודקת האם כל המשבצות בלוח תפוסות.

**יעילות הפונקציה:** (1)O - פעולות ביט בודדות.

**תיאור פסיאודו קוד של הפונקציה:**

1. איחוד שני הלוחות באמצעות פעולת OR בינארי

2. הפיכת הביטים באמצעות פעולת NOT

3. החלת מסיכה על התוצאה כדי להתעלם מביטים מעבר לגודל הלוח

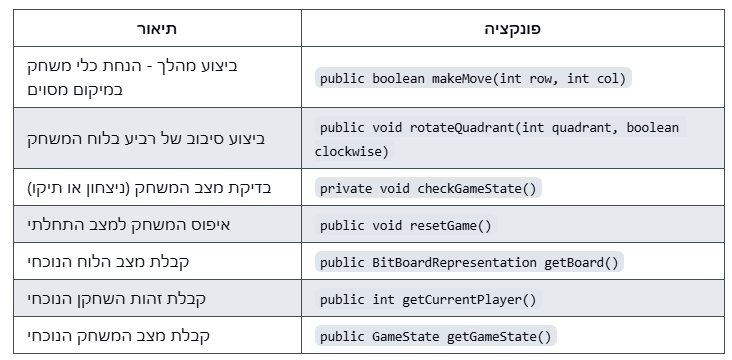
3.1. יצירת מסיכה עם מספר הביטים המדויק לייצוג הלוח

4. בדיקה האם התוצאה היא 0 (אין משבצות פנויות)

5. החזרת תוצאת הבדיקה

**שם מחלקה: PentagoModel**

**תיאור מחלקה:** המחלקה PentagoModel מייצגת את מודל המשחק Pentago. היא מיישמת את לוגיקת המשחק ומנהלת את מצב המשחק, כולל ייצוג הלוח באמצעות BitBoardRepresentation, ניהול תור השחקנים, ביצוע מהלכים (הנחת כלים וסיבוב רביעים), בדיקת מצב המשחק (ניצחון או תיקו) ואיפוס המשחק. המחלקה יורשת מ-Observable כדי לאפשר עדכון אוטומטי של התצוגה בכל שינוי במודל.



**public boolean makeMove(int row, int col):**

**הפרמטרים:** מספר השורה (0-5) מספר העמודה (0-5).

**מה הפונקציה מחזירה:** מחזירה true אם המהלך בוצע בהצלחה, ו-false אם המהלך אינו חוקי.

**מה הפונקציה מבצעת:** מבצעת מהלך של הנחת כלי משחק במיקום המבוקש אם המיקום פנוי.

**יעילות הפונקציה:** (1)O - פעולות פשוטות בלבד.

**תיאור פסיאודו קוד של הפונקציה:**

1. חישוב המיקום המלא בלוח

1.1. חשב מיקום = שורה \* 6 + עמודה

2. בדיקת תקינות המהלך

2.1. אם המיקום אינו פנוי

2.1.1. החזר שקר (מהלך לא חוקי)

3. הנחת כלי המשחק

3.1. הנח כלי של השחקן הנוכחי במיקום המבוקש

3.2. שמור את המיקום של המהלך האחרון

4. עדכון צופים

4.1. סמן שחל שינוי במודל

4.2. הודע לצופים על השינוי עם המידע "PIECE\_PLACED"

5. החזר אמת (מהלך בוצע בהצלחה)

**public void rotateQuadrant(int quadrant, boolean clockwise):**

**הפרמטרים:** מספר הרביע לסיבוב וכיוון הסיבוב (true לכיוון השעון, false נגד כיוון השעון).

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מסובבת רביע של הלוח, בודקת אם יש ניצחון ומחליפה שחקנים אם המשחק ממשיך.

**יעילות הפונקציה:** (1)O - מספר קבוע של פעולות.

**תיאור פסיאודו קוד של הפונקציה:**

1. סיבוב הרביע

1.1. הפעל את פונקצית סיבוב הרביע של הלוח

1.2. שמור את הרביע האחרון שסובב

1.3. שמור את כיוון הסיבוב האחרון

2. בדיקת מצב המשחק

2.1. בדוק אם יש ניצחון או תיקו

3. עדכון תור

3.1. אם המשחק עדיין בתהליך

3.1.1. החלף את השחקן הנוכחי (0 ל-1 או 1 ל-0)

4. עדכון צופים

4.1. סמן שחל שינוי במודל

4.2. הודע לצופים על השינוי עם המידע "BOARD\_UPDATED"

**private void checkGameState():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** בודקת אם יש ניצחון לאחד השחקנים או תיקו, ומעדכנת את מצב המשחק בהתאם.

**יעילות הפונקציה:** (1)O - הבדיקות נעשות בזמן קבוע.

**תיאור פסיאודו קוד של הפונקציה:**

1. בדיקת ניצחון לשחקן השחור

1.1. אם יש לשחקן השחור רצף מנצח

1.1.1. עדכן את מצב המשחק ל"ניצחון שחור"

2. בדיקת ניצחון לשחקן הלבן

2.1. אם יש לשחקן הלבן רצף מנצח

2.1.1. עדכן את מצב המשחק ל"ניצחון לבן"

3. בדיקת תיקו

3.1. אם הלוח מלא (ואין ניצחון לאף שחקן)

3.1.1. עדכן את מצב המשחק ל"תיקו"

**public void resetGame():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מאפסת את המשחק למצב התחלתי - לוח ריק, תור השחקן השחור ומצב משחק "בתהליך".

**יעילות הפונקציה:** (1)O - פעולות פשוטות בלבד.

**תיאור פסיאודו קוד של הפונקציה:**

1. איפוס הלוח

1.1. צור לוח חדש וריק

2. איפוס מצב המשחק

2.1. קבע את השחקן הנוכחי ל-0 (שחור)

2.2. קבע את מצב המשחק ל"בתהליך"

3. עדכון צופים

3.1. סמן שחל שינוי במודל

3.2. הודע לצופים על השינוי עם המידע "GAME\_RESET"

**public BitBoardRepresentation getBoard():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** מחזירה את מצב לוח המשחק הנוכחי.

**מה הפונקציה מבצעת:** מחזירה גישה למצב הלוח הנוכחי.

**יעילות הפונקציה:** (1)O - החזרת הפנייה בלבד.

**תיאור פסיאודו קוד של הפונקציה:**

1. החזרת הלוח

1.1. החזר הפנייה למצב הלוח הנוכחי

**public int getCurrentPlayer():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** מחזירה את מספר השחקן הנוכחי.

**מה הפונקציה מבצעת:** מחזירה את זהות השחקן שתורו לשחק.

**יעילות הפונקציה:** (1)O - החזרת ערך פשוט.

**תיאור פסיאודו קוד של הפונקציה:**

1. החזרת השחקן הנוכחי

1.1. החזר את מספר השחקן הנוכחי

**public GameState getGameState():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** מחזירה את מצב המשחק הנוכחי (בתהליך, ניצחון שחור, ניצחון אדום, תיקו).

**מה הפונקציה מבצעת:** מחזירה את מצב המשחק הנוכחי.

**יעילות הפונקציה:** (1)O - החזרת ערך פשוט.

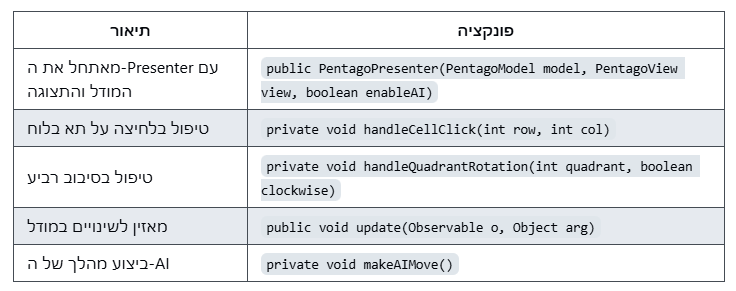
**תיאור פסיאודו קוד של הפונקציה:**

1. החזרת מצב המשחק

1.1. החזר את מצב המשחק הנוכחי

**שם מחלקה: PentagoPresenter**

**תיאור מחלקה:** המחלקה PentagoPresenter מיישמת את תבנית העיצוב Presenter במודל MVP. היא משמשת כמתווך בין המודל (PentagoModel) והתצוגה (PentagoView), מטפלת בלוגיקת המשחק, מגיבה לאירועי משתמש מהתצוגה, מעדכנת את המודל בהתאם, ומעדכנת את התצוגה עם שינויים במודל.



**public PentagoPresenter(PentagoModel model, PentagoView view, boolean enableAI):**

**הפרמטרים:** model (PentagoModel): המודל של המשחק,view (PentagoView): התצוגה של המשחק,

enableAI (boolean): האם להפעיל AI במשחק.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מאתחלת את ה-Presenter, מקשרת בין המודל והתצוגה, מגדירה מאזינים לאירועים, ומפעילה את ה-AI אם צריך.

**יעילות הפונקציה:** (1)O - פעולות אתחול בסיסיות.

**תיאור פסיאודו קוד של הפונקציה:**

1. אתחול הפרמטרים הבסיסיים

1.1. שמור את המודל

1.2. שמור את התצוגה

1.3. שמור את מצב הפעלת ה-AI

2. אם ה-AI מופעל

2.1. יצור מופע חדש של ה-AI עם המודל

2.2. הגדר את ה-AI לשחק כשחקן מספר 1 (לבן/אדום)

2.3. יצור מאגר חוטים יחיד עבור ה-AI

3. רישום המחלקה כצופה במודל

4. הגדרת מאזינים לאירועי המשתמש בתצוגה

5. עדכון ראשוני של התצוגה לפי המודל

6. אם ה-AI מופעל ותור השחקן הנוכחי הוא 1

6.1. הפעל את המהלך של ה-AI

**private void handleCellClick(int row, int col):**

**הפרמטרים:** מספר השורה ומספר העמודה.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מטפלת בלחיצה על תא בלוח - בודקת אם המהלך חוקי ומבצעת אותו במודל.

**יעילות הפונקציה:** (1)O - פעולות בדיקה ועדכון פשוטות.

**תיאור פסיאודו קוד של הפונקציה:**

1. בדיקת תקינות מצב המשחק

1.1. אם המשחק אינו במצב "IN\_PROGRESS"

1.1.1. צא מהפונקציה (אין מה לעשות)

2. בדיקת זכות תור במשחק עם AI

2.1. אם ה-AI מופעל והשחקן הנוכחי אינו 0 (לא תורו של האדם)

2.1.1. צא מהפונקציה (אין מה לעשות)

3. ביצוע המהלך

3.1. בצע את המהלך במודל עם השורה והעמודה שנבחרו

3.2. אם המהלך הצליח

3.2.1. עדכן את שלב המשחק בתצוגה למצב סיבוב רביע

**private void handleQuadrantRotation(int quadrant, boolean clockwise):**

**הפרמטרים:** מספר הרביע לסיבוב וכיוון הסיבוב (true לכיוון השעון, false נגד כיוון השעון).

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מטפלת בסיבוב רביע - בודקת אם המהלך חוקי, מבצעת אותו במודל ומפעילה את ה-AI אם צריך.

**יעילות הפונקציה:** (1)O - פעולות בדיקה ועדכון פשוטות.

**תיאור פסיאודו קוד של הפונקציה:**

1. בדיקת תקינות מצב המשחק

1.1. אם המשחק אינו במצב "IN\_PROGRESS"

1.1.1. צא מהפונקציה (אין מה לעשות)

2. בדיקת זכות תור במשחק עם AI

2.1. אם ה-AI מופעל והשחקן הנוכחי אינו 0 (לא תורו של האדם)

2.1.1. צא מהפונקציה (אין מה לעשות)

3. ביצוע סיבוב הרביע

3.1. בצע את סיבוב הרביע במודל

4. בדיקה אם צריך להפעיל את ה-AI

4.1. אם ה-AI מופעל, המשחק במצב "IN\_PROGRESS" והשחקן הנוכחי הוא 1

4.1.1. בצע מהלך של ה-AI

**public void update(Observable o, Object arg):**

**הפרמטרים:** האובייקט הנצפה שעדכן את הצופה וארגומנט נוסף שמספק מידע על העדכון.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מגיבה לעדכונים מהמודל ומעדכנת את התצוגה בהתאם.

**יעילות הפונקציה:** O(n²) - בגלל קריאה ל-updateViewFromModel.

**תיאור פסיאודו קוד של הפונקציה:**

1. בדיקה אם הארגומנט הוא מחרוזת

1.1. אם כן, המר את הארגומנט למחרוזת

1.2. בדוק את סוג העדכון

1.2.1. אם העדכון הוא "PIECE\_PLACED" (הנחת כלי)

1.2.1.1. עדכן את התצוגה לפי המודל

1.2.2. אם העדכון הוא "BOARD\_UPDATED" (עדכון לוח)

1.2.2.1. הגדר את שלב המשחק בתצוגה ל"PLACE\_PIECE"

1.2.2.2. עדכן את התצוגה לפי המודל

1.2.3. אם העדכון הוא "GAME\_RESET" (איפוס משחק)

1.2.3.1. הגדר את שלב המשחק בתצוגה ל"PLACE\_PIECE"

1.2.3.2. עדכן את התצוגה לפי המודל

**private void makeAIMove():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מפעילה את הבינה המלאכותית לביצוע מהלך מלא (הנחת כלי + סיבוב רביע).

**יעילות הפונקציה:** O(1) - פעולות מקבילות עם המתנה קבועה.

**תיאור פסיאודו קוד של הפונקציה:**

1. בדיקת תנאים מקדימים

1.1. אם ה-AI לא מופעל או המשחק אינו במצב "IN\_PROGRESS" או השחקן הנוכחי אינו 1

1.1.1. צא מהפונקציה (אין מה לעשות)

2. סימון שה-AI חושב כרגע

3. עדכון ממשק המשתמש באופן אסינכרוני

3.1. עדכון סטטוס המשחק ל"Computer is thinking..."

3.2. הסתרת כפתורי הסיבוב

4. הפעלת תהליך נפרד לביצוע מהלך ה-AI

4.1. המתנה קצרה (800 מילישניות)

4.2. עדכון המודל בבינה המלאכותית

4.3. קבלת מהלך (הנחת כלי) מהבינה המלאכותית

4.4. עדכון ממשק המשתמש באופן אסינכרוני

4.4.1. הנחת הכלי על הלוח במודל

4.4.2. עדכון שלב המשחק לסיבוב רביע

4.4.3. עדכון סטטוס המשחק ל"Computer is rotating..."

4.4.4. וידוא שכפתורי הסיבוב נשארים מוסתרים

4.5. המתנה נוספת (1200 מילישניות)

4.6. קבלת מהלך סיבוב מהבינה המלאכותית

4.7. עדכון ממשק המשתמש באופן אסינכרוני

4.7.1. הפעלת אנימציה של סיבוב בלי להציג את הכפתורים

4.7.2. אחרי האנימציה, בצע את הסיבוב במודל

4.7.3. סמן שה-AI סיים לחשוב

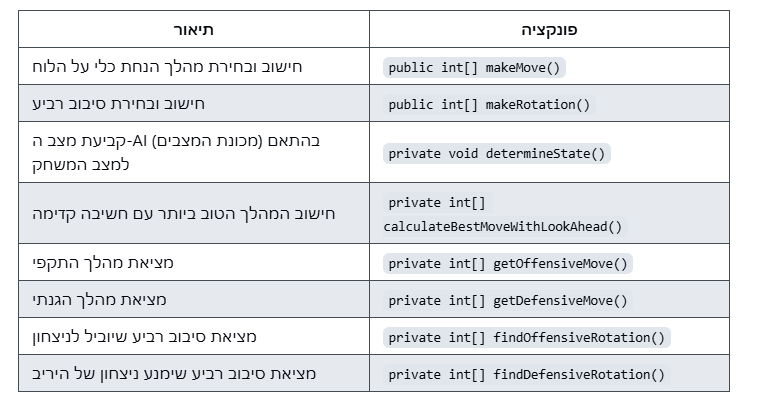
4.8. טיפול בשגיאות

4.8.1. הדפסת מידע על השגיאה

4.8.2. סימון שה-AI סיים לחשוב

**שם מחלקה: PentagoAI**

**תיאור מחלקה:** המחלקה PentagoAI מיישמת בינה מלאכותית לשחק במשחק Pentago. היא מבוססת על מכונת מצבים סופית (FSM - Finite State Machine) המחליטה על אסטרטגיית המשחק ומבצעת מהלכים אינטליגנטיים. המחלקה תומכת במצבי משחק שונים כמו התקפה, הגנה, שליטה במרכז, חסימת פינות, בניית דפוסים אסטרטגיים וחשיבה קדימה. המחלקה משלבת הערכה של מצב הלוח, אסטרטגיות משחק מתקדמות, וגם אלמנט מסוים של אקראיות כדי שהמשחק לא יהיה צפוי מדי.



**public int[] makeMove():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** מערך בגודל 2 המייצג את השורה והעמודה של המהלך הנבחר.

**מה הפונקציה מבצעת:** מחשבת ובוחרת את המהלך הטוב ביותר להנחת כלי על הלוח בהתאם למצב המשחק ולמצב ה-AI.

**יעילות הפונקציה:** O(n²) - סריקת כל אפשרויות המהלכים וחישוב הערכה.

**תיאור פסיאודו קוד של הפונקציה:**

1. הגדלת מונה המהלכים

2. קביעת מצב ה-AI לפני קבלת החלטה

3. החלטה על שימוש באסטרטגיית חיפוש מתקדמת

3.1. אם נבחרה אסטרטגיה מתקדמת (הסיכוי לכך הוא 80%) ומצב ה-AI מתאים

3.1.1. חישוב המהלך הטוב ביותר עם חשיבה קדימה

3.2. אחרת, בחירת מהלך בהתאם למצב ה-AI הנוכחי

3.2.1. אם במצב התקפה, בחר מהלך התקפי

3.2.2. אם במצב הגנה, בחר מהלך הגנתי

3.2.3. אם במצב שליטה במרכז, בחר מהלך אסטרטגי

3.2.4. אם במצב שליטה בקצוות, בחר מהלך לשליטה בקצוות

3.2.5. אם במצב חסימת פינות, בחר מהלך לחסימת פינות

3.2.6. אם במצב בניית דפוס, בחר מהלך לבניית דפוס מתקדם

3.2.7. אם במצב שליטה בסיבוב, בחר מהלך המכין לסיבוב אסטרטגי

3.2.8. אחרת, בחר מהלך אקראי

4. החזרת המהלך הנבחר

**public int[] makeRotation():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** מערך בגודל 2 המייצג את הרביע לסיבוב ואת כיוון הסיבוב (1 לכיוון השעון, 0 נגד כיוון השעון).

**מה הפונקציה מבצעת:** מחשבת ובוחרת את סיבוב הרביע הטוב ביותר בהתאם למצב המשחק.

**יעילות הפונקציה:** O(n²) - בדיקת כל אפשרויות הסיבוב וחישוב הערכה.

**תיאור פסיאודו קוד של הפונקציה:**

1. החלטה על ניתוח מעמיק (הסיכוי לכך הוא 80%)

2. אם נבחר ניתוח מעמיק

2.1. בדיקה אם יש מהלך סיבוב התקפי

2.1.1. אם כן, בחר אותו

2.2. אחרת, בדיקה אם יש מהלך סיבוב הגנתי

2.2.1. אם כן, בחר אותו

2.3. אחרת, בדיקה אם יש מהלך סיבוב אסטרטגי משופר

2.3.1. אם כן, בחר אותו

2.4. אחרת, בחר סיבוב חכם אקראי

3. אחרת (לוגיקה פשוטה יותר)

3.1. בדיקה אם יש מהלך סיבוב התקפי

3.1.1. אם כן, בחר אותו

3.2. אחרת, בדיקה אם יש מהלך סיבוב הגנתי

3.2.1. אם כן, בחר אותו

3.3. אחרת, בחר סיבוב אקראי

4. החזרת הרביע והכיוון שנבחרו

**private void determineState():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** קובעת את מצב ה-AI (אסטרטגיית המשחק) בהתאם למצב הלוח הנוכחי.

**יעילות הפונקציה:** O(n²) - בדיקות מצב הלוח דורשות סריקה.

**תיאור פסיאודו קוד של הפונקציה:**

1. בדיקה אם יש אפשרות לניצחון מיידי

1.1. אם כן, עבור למצב התקפה ויציאה

2. בדיקה אם היריב קרוב לניצחון

2.1. אם כן, עבור למצב הגנה ויציאה

3. בדיקה אם יש צורך בחישוב מהלכים קדימה

3.1. אם כן, עבור למצב חשיבה קדימה ויציאה

4. אם המשחק בתחילתו (עד 4 מהלכים)

4.1. בדיקה אם יש הזדמנות לבניית דפוס אסטרטגי

4.1.1. אם כן, עבור למצב בניית דפוס ויציאה

5. בדיקה אם יש עמדות אסטרטגיות פנויות במרכז

5.1. אם כן, עבור למצב שליטה במרכז ויציאה

6. בדיקה אם יש איום על פינות חשובות

6.1. אם כן, עבור למצב חסימת פינות ויציאה

7. בדיקה אם יש הזדמנות לבניית דפוס אסטרטגי

7.1. אם כן, עבור למצב בניית דפוס ויציאה

8. בדיקה אם כדאי להתמקד בשליטה בקצוות

8.1. אם כן, עבור למצב שליטה בקצוות ויציאה

9. אם המשחק בשלב מתקדם (מעל 8 מהלכים)

9.1. עבור למצב שליטה בסיבוב ויציאה

10. ברירת מחדל - בחירה אקראית בין בניית דפוס לשליטה במרכז

**private int[] calculateBestMoveWithLookAhead():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** מערך בגודל 2 המייצג את המיקום הטוב ביותר להנחת כלי.

**מה הפונקציה מבצעת:** מחשבת את המהלך הטוב ביותר תוך חשיבה קדימה וניתוח של מצבי משחק עתידיים.

**יעילות הפונקציה:** O(n³) - בדיקת כל המהלכים האפשריים וחישוב ציון מורכב לכל אחד.

**תיאור פסיאודו קוד של הפונקציה:**

1. קבלת רשימת כל המהלכים האפשריים

2. אתחול משתנים למהלך הטוב ביותר והציון הטוב ביותר

3. עבור כל מהלך אפשרי

3.1. חישוב ציון המהלך

3.2. אם הציון גבוה יותר מהציון הטוב ביותר עד כה

3.2.1. עדכון המהלך והציון הטובים ביותר

4. אם לא נמצא מהלך טוב, בחר מהלך אסטרטגי

5. החזרת המהלך הטוב ביותר שנמצא

**private int[] getOffensiveMove():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** מערך בגודל 2 המייצג את המיקום הטוב ביותר למהלך התקפי.

**מה הפונקציה מבצעת:** מחפשת מהלך שיכול ליצור רצף מנצח או להתקרב לניצחון.

**יעילות הפונקציה:** O(n²) - סריקת הלוח למציאת רצפים פוטנציאליים.

**תיאור פסיאודו קוד של הפונקציה:**

1. חיפוש מהלכים שמובילים לרצף של 4 כלים

1.1. אם נמצאו מהלכים כאלה

1.1.1. בחר באקראי מהלך מהמהלכים שנמצאו והחזר אותו

2. חיפוש מהלכים שמובילים לרצף של 3 כלים

2.1. אם נמצאו מהלכים כאלה

2.1.1. בחר באקראי מהלך מהמהלכים שנמצאו והחזר אותו

3. אם לא נמצאו אפשרויות התקפיות ברורות

3.1. חזרה למהלך אסטרטגי

**private int[] getDefensiveMove():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** מערך בגודל 2 המייצג את המיקום הטוב ביותר למהלך הגנתי.

**מה הפונקציה מבצעת:** מחפשת מהלך שיכול למנוע מהיריב ליצור רצף מנצח או להתקרב לניצחון.

**יעילות הפונקציה:** O(n²) - סריקת הלוח למציאת רצפים פוטנציאליים של היריב.

**תיאור פסיאודו קוד של הפונקציה:**

1. חיפוש מהלכים שחוסמים רצף של 4 כלים של היריב

1.1. אם נמצאו מהלכים כאלה

1.1.1. בחר באקראי מהלך מהמהלכים שנמצאו והחזר אותו

2. חיפוש מהלכים שחוסמים רצף של 3 כלים של היריב

2.1. אם נמצאו מהלכים כאלה

2.1.1. בחר באקראי מהלך מהמהלכים שנמצאו והחזר אותו

3. אם לא נמצאו מהלכי הגנה דחופים

3.1. חזרה למהלך אסטרטגי

**private int[] findOffensiveRotation():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** מערך בגודל 2 המייצג את הרביע והכיוון לסיבוב שיוביל לניצחון, או null אם לא נמצא.

**מה הפונקציה מבצעת:** מחפשת סיבוב רביע שיוביל לניצחון מיידי.

**יעילות הפונקציה:** O(1) - בדיקת מספר קבוע של אפשרויות סיבוב (8 אפשרויות).

**תיאור פסיאודו קוד של הפונקציה:**

1. עבור כל רביע מ-0 עד 3

1.1. עבור כל כיוון (0 - נגד כיוון השעון, 1 - עם כיוון השעון)

1.1.1. יצירת מודל זמני לבדיקת המהלך

1.1.2. סימולציה של סיבוב הרביע

1.1.3. בדיקה אם הסיבוב יוצר רצף מנצח

1.1.3.1. אם כן, החזר את הרביע והכיוון

2. אם לא נמצא סיבוב מנצח, החזר null

**private int[] findDefensiveRotation():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** מערך בגודל 2 המייצג את הרביע והכיוון לסיבוב שימנע ניצחון של היריב, או null אם לא נמצא.

**מה הפונקציה מבצעת:** מחפשת סיבוב רביע שימנע מהיריב לנצח במהלך הבא.

**יעילות הפונקציה:** O(1) - בדיקת מספר קבוע של אפשרויות סיבוב (8 אפשרויות).

**תיאור פסיאודו קוד של הפונקציה:**

1. עבור כל רביע מ-0 עד 3

1.1. עבור כל כיוון (0 - נגד כיוון השעון, 1 - עם כיוון השעון)

1.1.1. יצירת מודל זמני לבדיקת המהלך

1.1.2. סימולציה של סיבוב הרביע

1.1.3. בדיקה אם הסיבוב מונע ניצחון של היריב

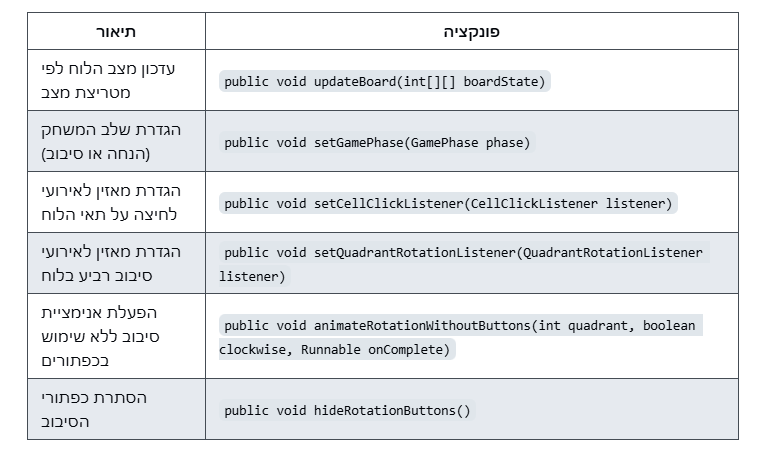
1.1.4. בדיקה אם הסיבוב מקטין את מספר האיומים של היריב

1.1.4.1. אם שתי התנאים מתקיימים, החזר את הרביע והכיוון

2. אם לא נמצא סיבוב הגנתי, החזר null

**שם מחלקה: BoardView**

**תיאור מחלקה:** המחלקה BoardView מייצגת את התצוגה הגרפית של לוח משחק Pentago. היא אחראית על הצגה ותצוגה של לוח המשחק 6X6 המחולק ל-4 רביעים בגודל 3X3, כולל הוספת כפתורי סיבוב לכל רביע, אנימציות סיבוב, וניהול התנהגות אינטראקטיבית של הלוח. המחלקה מיישמת את ממשק המשתמש במשחק, מציגה את הכלים השחורים והאדומים, ומטפלת באירועי מקלדת ועכבר של המשתמש.



**public void updateBoard(int[][] boardState):**

**הפרמטרים:** מטריצה 6X6 המייצגת את מצב הלוח הנוכחי, כאשר -1 מייצג משבצת ריקה, 0 מייצג כלי שחור ו-1 מייצג כלי אדום.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מעדכנת את התצוגה הגרפית של הלוח בהתאם למצב הלוח שהתקבל כפרמטר.

**יעילות הפונקציה:** O(n²) - עבור לוח גודל n×n.

**תיאור פסיאודו קוד של הפונקציה:**

1. עבור כל שורה (r) מ-0 עד 5

1.1. עבור כל עמודה (c) מ-0 עד 5

1.1.1. חישוב מספר הרביע לפי השורה והעמודה: (r / 3) \* 2 + (c / 3)

1.1.2. חישוב השורה המקומית בתוך הרביע: r % 3

1.1.3. חישוב העמודה המקומית בתוך הרביע: c % 3

1.1.4. עדכון המשבצת המתאימה ברביע המתאים עם הערך מ-boardState[r][c]

**public void setGamePhase(GamePhase phase):**

**הפרמטרים:** שלב המשחק הנוכחי (PLACE\_PIECE, ROTATE\_QUADRANT, GAME\_OVER).

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מעדכנת את שלב המשחק הנוכחי ומשנה את התנהגות הלוח בהתאם (למשל מציגה או מסתירה כפתורי סיבוב).

**יעילות הפונקציה:** O(n) - עבור n רביעים (כאן n=4).

**תיאור פסיאודו קוד של הפונקציה:**

1. שמירת שלב המשחק החדש

2. עבור כל רביע בלוח

2.1. עדכון שלב המשחק של הרביע

3. עדכון נראות כפתורי הסיבוב בהתאם לשלב החדש

**public void setCellClickListener(CellClickListener listener):**

**הפרמטרים:** מאזין לאירועי לחיצה על תאי הלוח.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מגדירה מאזין שיטפל באירועי לחיצה על תאי הלוח בכל רביעיו.

**יעילות הפונקציה:** O(n) - עבור n רביעים (כאן n=4).

**תיאור פסיאודו קוד של הפונקציה:**

1. עבור כל רביע בלוח

1.1. הגדרת מאזין הלחיצות עבור הרביע

**public void setQuadrantRotationListener(QuadrantRotationListener listener):**

**הפרמטרים:** מאזין לאירועי לחיצה על תאי הלוח.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מגדירה מאזין שיטפל באירועי סיבוב של רביעי הלוח.

**יעילות הפונקציה:** O(n) - עבור n רביעים (כאן n=4).

**תיאור פסיאודו קוד של הפונקציה:**

1. עבור כל רביע בלוח

1.1. הגדרת מאזין סיבוב הרביע עבור הרביע

**public void animateRotationWithoutButtons(int quadrant, boolean clockwise, Runnable onComplete):**

**הפרמטרים:** מספר הרביע לסיבוב (0-3), כיוון הסיבוב (true לכיוון השעון, false נגד כיוון השעון), פעולה לביצוע בסיום האנימציה.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מפעילה אנימציית סיבוב אוטומטית (ללא לחיצה על כפתור) עבור רביע מסוים, לרוב משמשת למהלכי ה-AI.

**יעילות הפונקציה:** O(1) - פעולות פשוטות.

**תיאור פסיאודו קוד של הפונקציה:**

1. הסתרת כל כפתורי הסיבוב של הלוח

2. הפעלת אנימציית סיבוב על הרביע המבוקש בכיוון שנבחר, עם פעולת השלמה שתבוצע בסיום האנימציה

**public void hideRotationButtons():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מסתירה את כל כפתורי הסיבוב על הלוח.

**יעילות הפונקציה:** O(n) - עבור n כפתורים (כאן n=8).

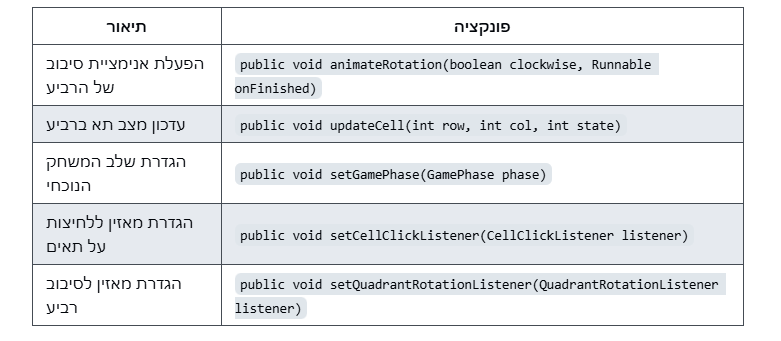
**תיאור פסיאודו קוד של הפונקציה:**

1. עבור כל כפתור סיבוב בלוח

1.1. הגדר את נראות הכפתור לשקר (הסתר)

**שם מחלקה: QuadrantView**

**תיאור מחלקה:** המחלקה QuadrantView מייצגת רביע אחד מתוך ארבעת הרביעים המרכיבים את לוח משחק Pentago. כל רביע הוא לוח משנה בגודל 3X3 המכיל תאי משחק המיוצגים כעיגולים. המחלקה אחראית על תצוגה גרפית של הרביע, טיפול בלחיצות משתמש על תאים, ביצוע אנימציות סיבוב, ועדכון מצב התאים בהתאם למצב המשחק. היא מיישמת את עיצוב הלוח בצבעי עץ ומכילה הגדרות לאנימציות סיבוב הרביע.



**public void animateRotation(boolean clockwise, Runnable onFinished):**

**הפרמטרים:** האם לסובב את הרביע בכיוון השעון (true) או נגד כיוון השעון (false),פעולה שתבוצע בסיום האנימציה.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מפעילה אנימציית סיבוב של הרביע ב-90 מעלות בכיוון המבוקש, ומבצעת פעולה מוגדרת בסיום האנימציה.

**יעילות הפונקציה:** O(1) - פעולות פשוטות להגדרת והפעלת אנימציה.

**תיאור פסיאודו קוד של הפונקציה:**

1. יצירת אנימציית סיבוב עבור הרביע למשך חצי שנייה

2. אם הבקשה היא לסיבוב בכיוון השעון

2.1. הגדר את זווית הסיבוב ל-90 מעלות

3. אחרת (סיבוב נגד כיוון השעון)

3.1. הגדר את זווית הסיבוב ל-90 מעלות בכיוון השלילי (-90)

4. הגדר פעולה שתתבצע בסיום האנימציה

4.1. איפוס זווית הסיבוב של הרכיב הגרפי ל-0

4.2. אם הועברה פעולת סיום

4.2.1. הפעל את פעולת הסיום

5. הפעל את אנימציית הסיבוב

**public void updateCell(int row, int col, int state):**

**הפרמטרים:** מספר השורה של התא (0-2) מספר העמודה של התא (0-2) מצב התא (-1: ריק, 0: שחור, 1: אדום).

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מעדכנת את המצב הגרפי של תא ספציפי ברביע לפי המצב שהתקבל.

**יעילות הפונקציה:** O(1) - עדכון תא בודד.

**תיאור פסיאודו קוד של הפונקציה:**

1. קבלת העיגול המייצג את התא במיקום המבוקש

2. בדיקת מצב התא

2.1. אם המצב הוא -1 (ריק)

2.1.1. צבע את העיגול בצבע ריק חצי שקוף

2.2. אם המצב הוא 0 (שחור)

2.2.1. צבע את העיגול בצבע שחור

2.3. אם המצב הוא 1 (אדום/לבן)

2.3.1. צבע את העיגול בצבע אדום

**public void setGamePhase(GamePhase phase):**

**הפרמטרים:** שלב המשחק החדש (PLACE\_PIECE, ROTATE\_QUADRANT, GAME\_OVER).

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מעדכנת את שלב המשחק הנוכחי של הרביע, משפיע על התנהגות הרביע בתגובה ללחיצות משתמש.

**יעילות הפונקציה:** O(1) - עדכון משתנה פשוט.

**תיאור פסיאודו קוד של הפונקציה:**

1. עדכון משתנה שלב המשחק הנוכחי לערך החדש שהתקבל

**public void setCellClickListener(CellClickListener listener):**

**הפרמטרים:** מאזין ללחיצות על תאים ברביע.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מגדירה מאזין שיטפל באירועי לחיצה על תאי הרביע.

**יעילות הפונקציה:** O(1) - הגדרת מאזין יחיד.

**תיאור פסיאודו קוד של הפונקציה:**

1. שמירת המאזין שהתקבל כמאזין לחיצות התאים של הרביע

**public void setQuadrantRotationListener(QuadrantRotationListener listener):**

**הפרמטרים:** מאזין לאירועי סיבוב של הרביע.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מגדירה מאזין שיטפל באירועי סיבוב של הרביע.

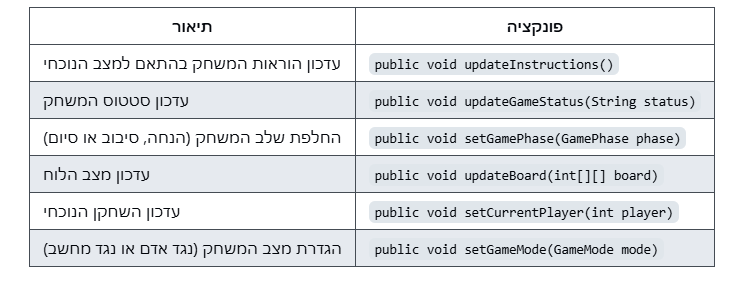
**יעילות הפונקציה:** O(1) - הגדרת מאזין יחיד.

**תיאור פסיאודו קוד של הפונקציה:**

1. שמירת המאזין שהתקבל כמאזין סיבוב הרביע

**שם מחלקה: PentagoView**

**תיאור מחלקה:** המחלקה PentagoView מייצגת את התצוגה המלאה של המשחק. היא מרכזת את כל רכיבי הממשק המשתמש כולל לוח המשחק (BoardView), תוויות מצב ומידע, וכפתורי משחק חדש וחזרה לתפריט. המחלקה אחראית על תיאום התצוגה בהתאם למצב המשחק הנוכחי, הצגת הוראות מתאימות לשחקן, ניהול מעברים בין שלבי המשחק השונים (הנחת כלי, סיבוב רביע, סיום משחק), ותמיכה במצבי משחק שונים (נגד אדם או נגד מחשב).



**public void updateInstructions():**

**הפרמטרים:** אין.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מעדכנת את הוראות המשחק המוצגות לשחקן בהתאם לשלב המשחק הנוכחי ולשחקן הנוכחי.

**יעילות הפונקציה:** O(1) - פעולות פשוטות לעדכון תווית.

**תיאור פסיאודו קוד של הפונקציה:**

1. קביעת שם השחקן הנוכחי:

1.1. אם מצב המשחק הוא נגד מחשב והשחקן הנוכחי הוא 1

1.1.1. שם השחקן יהיה "Computer"

1.2. אחרת, אם השחקן הנוכחי הוא 0

1.2.1. שם השחקן יהיה "Black"

1.3. אחרת

1.3.1. שם השחקן יהיה "Red"

2. בדיקת שלב המשחק הנוכחי:

2.1. אם השלב הוא הנחת כלי

2.1.1. הצג הודעה: "Player [שם השחקן], place a piece"

2.2. אם השלב הוא סיבוב רביע

2.2.1. הצג הודעה: "Player [שם השחקן], rotate a quadrant"

2.3. אם השלב הוא סיום המשחק

2.3.1. הצג הודעה: "Game Over! Press 'New Game' to restart"

**public void updateGameStatus(String status):**

**הפרמטרים:** הסטטוס החדש שיוצג למשתמש.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מעדכנת את תווית הסטטוס המרכזית של המשחק עם טקסט חדש.

**יעילות הפונקציה:** O(1) - פעולות פשוטות לעדכון תווית.

**תיאור פסיאודו קוד של הפונקציה:**

1. עדכון טקסט תווית הסטטוס לטקסט שהתקבל כפרמטר

**public void setGamePhase(GamePhase phase):**

**הפרמטרים:** שלב המשחק החדש (PLACE\_PIECE, ROTATE\_QUADRANT, GAME\_OVER).

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מעדכנת את שלב המשחק הנוכחי, מעדכנת את תצוגת הלוח לשלב החדש, ומעדכנת את ההוראות המוצגות.

**יעילות הפונקציה:** O(1) - עדכון משתנים ותוויות.

**תיאור פסיאודו קוד של הפונקציה:**

1. שמירת שלב המשחק החדש

2. עדכון שלב המשחק בתצוגת הלוח

3. עדכון ההוראות המוצגות לשחקן בהתאם לשלב החדש

**public void updateBoard(int[][] board):**

**הפרמטרים:** מטריצה 6X6 המייצגת את מצב הלוח הנוכחי.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מעדכנת את תצוגת הלוח לפי מצב הלוח החדש.

**יעילות הפונקציה:** O(n²) - העברת מטריצת הלוח לתצוגת הלוח.

**תיאור פסיאודו קוד של הפונקציה:**

1. העברת מטריצת הלוח לתצוגת הלוח לעדכון

**public void setCurrentPlayer(int player):**

**הפרמטרים:** מספר השחקן הנוכחי (0 לשחור, 1 לאדום/מחשב).

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מעדכנת את השחקן הנוכחי בתצוגת הלוח ומעדכנת את ההוראות המוצגות.

**יעילות הפונקציה:** O(1) - עדכון משתנים ותוויות.

**תיאור פסיאודו קוד של הפונקציה:**

1. עדכון השחקן הנוכחי בתצוגת הלוח

2. עדכון ההוראות המוצגות לשחקן בהתאם לשחקן החדש

**public void setGameMode(GameMode mode):**

**הפרמטרים:** מצב המשחק החדש (HUMAN לשחקן אנושי, AI למשחק נגד מחשב).

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מעדכנת את מצב המשחק (נגד אדם או נגד מחשב) ומעדכנת את ההוראות המוצגות.

**יעילות הפונקציה:** O(1) - עדכון משתנים ותוויות.

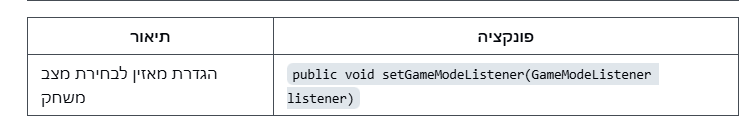
**תיאור פסיאודו קוד של הפונקציה:**

1. שמירת מצב המשחק החדש

2. עדכון ההוראות המוצגות לשחקן בהתאם למצב החדש

**שם מחלקה: MainMenuView**

**תיאור מחלקה:** המחלקה MainMenuView מייצגת את התפריט הראשי של משחק Pentago. היא מספקת ממשק משתמש גרפי המאפשר לשחקן לבחור את מצב המשחק הרצוי - משחק נגד שחקן אחר או משחק נגד המחשב (AI). המחלקה מיישמת עיצוב מושך לתפריט הראשי, הכולל כותרת ראשית, כפתורים מעוצבים עם אפקט מעבר עכבר, ורקע מותאם.



**public void setGameModeListener(GameModeListener listener):**

**הפרמטרים:** מאזין שיטפל באירוע בחירת מצב משחק.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** מגדירה מאזין שיקבל התראה כאשר המשתמש בוחר מצב משחק (נגד שחקן או נגד מחשב).

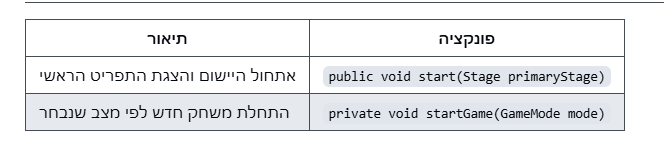
**יעילות הפונקציה:** O(1) - פעולה פשוטה של הקצאת מאזין.

**תיאור פסיאודו קוד של הפונקציה:**

1. שמירת המאזין שהתקבל כמאזין למצב המשחק הנבחר

**שם מחלקה: PentagoApp**

**תיאור מחלקה:** המחלקה PentagoApp היא נקודת הכניסה העיקרית לתוכנת המשחק Pentago. היא יורשת מהמחלקה Application של JavaFX ומיישמת את ארכיטקטורת Model-View-Presenter (MVP) ליישום המשחק. המחלקה אחראית על אתחול חלון המשחק, יצירת התפריט הראשי, ניהול החלפת סצנות בין תפריט המשחק לבין משחק פעיל, ואתחול כל הרכיבים הדרושים למשחק בהתאם למצב המשחק שנבחר (משחק נגד שחקן או נגד המחשב).



**public void start(Stage primaryStage):**

**הפרמטרים:** חלון המשחק הראשי שמועבר על ידי מנגנון JavaFX.

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** אתחול היישום על ידי יצירת תפריט ראשי, הגדרת סצנה ראשונה, והצגת חלון המשחק למשתמש.

**יעילות הפונקציה:** O(1) - פעולות אתחול קבועות.

**תיאור פסיאודו קוד של הפונקציה:**

1. שמירת חלון המשחק הראשי לשימוש בהמשך

2. יצירת תצוגת תפריט ראשי

3. הגדרת מאזין לבחירת מצב משחק שיקרא לפונקציה startGame עם המצב שנבחר

4. יצירת סצנת תפריט עם תצוגת התפריט בגודל 800x700

5. טעינת גיליון סגנונות CSS לסצנה

6. הגדרת כותרת לחלון המשחק

7. הגדרת סצנת התפריט כסצנה הנוכחית בחלון המשחק

8. הגדרת החלון כלא ניתן לשינוי גודל

9. הצגת חלון המשחק

**private void startGame(GameMode mode):**

**הפרמטרים:** מצב המשחק שנבחר (PLAYER\_VS\_PLAYER או PLAYER\_VS\_AI).

**מה הפונקציה מחזירה:** אין ערך חזרה.

**מה הפונקציה מבצעת:** יוצרת משחק חדש בהתאם למצב שנבחר, כולל אתחול המודל, התצוגה והמציג (presenter), ועוברת מהתפריט למסך המשחק.

**יעילות הפונקציה:** O(1) - פעולות אתחול קבועות.

**תיאור פסיאודו קוד של הפונקציה:**

1. יצירת מודל משחק חדש

2. יצירת תצוגת משחק חדשה

3. יצירת מציג משחק חדש המקשר בין המודל לתצוגה, עם הגדרת מצב AI לפי הבחירה

4. הגדרת פעולה לכפתור החזרה לתפריט:

4.1. עצירת ה-AI אם הוא פעיל

4.2. החלפת הסצנה לסצנת התפריט

4.3. עדכון כותרת החלון

5. בדיקת מצב המשחק שנבחר:

5.1. אם נבחר מצב משחק נגד מחשב

5.1.1. הגדרת מצב המשחק ל-AI בתצוגה

5.1.2. עדכון כותרת הסטטוס ל-"Player VS Computer"

5.2. אחרת (נבחר מצב משחק נגד שחקן)

5.2.1. הגדרת מצב המשחק ל-HUMAN בתצוגה

5.2.2. עדכון כותרת הסטטוס ל-"Player VS Player"

6. יצירת סצנת משחק חדשה עם תצוגת המשחק בגודל 800x700

7. טעינת גיליון סגנונות CSS לסצנה

8. החלפת הסצנה הנוכחית בחלון לסצנת המשחק

9. עדכון כותרת החלון ל-"PENTAGO - Game in Progress"