*עבודת גמר*

*המשחק - Abalone*

**סמל מוסד:**471029

**שם מכללה:**מכללת אורט הרמלין נתניה

**שם הסטודנט:** דניאל מרדכי ליוש

**ת"ז הסטודנט:** 327885992  
**שמות המנחים:** אלון חיימוביץ ואורי וולטמן.  
**שם הפרויקט:** ***Abalone*תאריך הגשה:**



***תוכן עניינים***

[תקציר – 3](#_Toc163653634)

[מטרת הפרויקט: 3](#_Toc163653635)

[**1.** **ניתוח המשחק:** 3](#_Toc163653636)

[**2.** **תכנון ופיתוח:** 3](#_Toc163653637)

[**3.** **בדיקות:** 3](#_Toc163653638)

[**4.** **שיפור ושדרוג:** 3](#_Toc163653639)

[מושגים – 4](#_Toc163653640)

[תיאור הנושא - 5](#_Toc163653641)

[רקע תיאורטי - 8](#_Toc163653642)

[תיאור הבעיה האלגוריתמית – 9](#_Toc163653643)

[סקירת אלגוריתמים בתחום הבעיה – 10](#_Toc163653644)

[אלגוריתם נבחר – 11](#_Toc163653645)

[Top-down Level Design - 12](#_Toc163653646)

[תיאור מילולי של ארכיטקטורת הפרויקט - 12](#_Toc163653647)

[UML Use cases - 12](#_Toc163653648)

[מבנה נתונים - 12](#_Toc163653649)

[תיאור סביבת העבודה ושפת התכנות – 12](#_Toc163653650)

[אלגוריתם ראשי (פסיאדו קוד) - 12](#_Toc163653651)

[תיאור API - 12](#_Toc163653652)

[UML Class Diagram - 12](#_Toc163653653)

[הפונקציות/המחלקות הראשיות בפרויקט - 12](#_Toc163653654)

[התוכנית הראשית שמנהלת את המשחק - 12](#_Toc163653655)

[מדריך למשתמש - 12](#_Toc163653656)

[רפלקציה - 12](#_Toc163653657)

[ביבליוגרפיה - 12](#_Toc163653658)

[נספחים - 13](#_Toc163653659)

***תקציר –***

*מטרת הפרויקט:*

אבאלון הוא משחק לוח אסטרטגי מופשט לשני שחקנים.   
שתוכנן על ידי מישל ללט ולורנט לוי בשנת 1987. אבאלון ייצא לשוק בשנת 1990 ונמכר במיליוני עותקים.  
משך זמן המשחק הוא בין 10 דקות לשעתיים!!!

פיתוח המשחק בשפת JAVA, ביחד עם סוכן AI המשתמש במערכת הכרעת מצבים לביצוע המהלכים שלו. למשחק יהיו כמה מצבי משחק שהמשתמש יוכל לבחור מהם, שחקן אנושי מול מחשב, שחקן אנושי מול שחקן אנושי ושחקן מחשב מול שחקן מחשב. הפרויקט יכלול אנימציות לשיפור חווית המשתמש.

*הליך העבודה:*

1. **ניתוח המשחק:**

* הבנת חוקי המשחק, תנאי הניצחון וההפסד, אפשרויות המהלכים והאסטרטגיות השונות.
* בחירת מערכת הכרעת המצבים המתאימה לפי הגדרות הפרויקט, שתאפשר קבלת החלטות יעילות עבור סוכן ה AI.

1. **תכנון ופיתוח:**

* תכנון ועיצוב ממשק משתמש גרפי בסיסי למשחק.
* פיתוח לוגיקת המשחק, תוך שימוש במבני נתונים מתאימים(כמו רשימות, מטריצות וטבלאות גיבוב) לייצוג מצב המשחק.
* מימוש מערכת הכרעת מצבים שבחרתי, תוך התאמה לחוקי המשחק ולתכונותיו.
* שילוב מערכת הכרעת המצבים שפיתחנו במשחק.

1. **בדיקות:**

* ביצוע בדיקות מקיפות לוודא תקינות המשחק, תפקוד סוכן הai והתנהגותו במצבים שונים.
* שיפור וחידוד אלגוריתם קבלת ההחלטות של סוכן ה AI על פי תוצאות הבדיקות.
* שיפור ביצועי המערכת לאחר הבדיקות.

1. **שיפור ושדרוג:**

* שדרוג ממשק המשתמש, מבחינת הנראות והנוחות.
* הוספת פיצ'רים לשיפור חווית המשתמש.
* הוספת מערכת למעקב וניתוח סטטיסטיקות של המשחקים.

*אתגרים:*

* תכנון ומימוש מערכת הכרעת מצבים יעילה.
* מימוש מבנה הנתונים לייצוג מצב המשחק ותנועה של שחקנים.

***מושגים –***

* **אבאלון** – משחק לוח אסטרטגי שנוצר בצרפת, מוכר בכל העולם, כלליו פשוטים והמשחק האסטרטגי שלו מורכב. במשחק יש שני שחקנים, כל אחד עם 14 גולות, מטרת המשחק היא לדחוף 6 גולות של היריב מחוץ ללוח.
* **סוכן AI** – תוכנה שתוכננה לשחק אבאלון נגיד שחקן אנושי או נגד סוכן AI אחר, סוכן הAI משתמש באלגוריתמים ופונקציות הערכה כדי לקבל החלטות במשחק. מטרתו היא למקסם את סיכויי הניצחון שלו על ידי ביצוע המהלך הטוב ביותר בכל תור.
* **פונקציות הערכה היוריסטיות** – פונקציות המעניקות ציון למצב לוח ספציפי. היוריסטיות עוזרות לשחקן המחשב(סוכן הAI) להחליט איזה מהלך יהיה הכי טוב לבצע עכשיו מבלי הצורך לחפש את כל עץ המשחק.
* **מהלכים בקו(In-Line Moves) -** מהלכים שבהם קבוצת גולות מתקדמת בכיוון אחד באופן ישיר.
* **מהלכי צד(Broadside Moves) –** מהלכים שבהם קבוצת גולות זזה לצד באופן רוחבי לכיוון התקדמותם.
* **מהלכי דחיפה (Sumito moves) –**

מהלכים בהם קבוצת גולות של שחקן דוחפת קבוצת גולות של היריב (קבוצה בת גולה אחת או שתיים לכל היותר).

* **אלגוריתמים -** הוראות מפורשות לביצוע משימה מסוימת.
* **JAVA -** שפת תכנות פופולרית, מונחת עצמים.
* **GUI –** ממשק משתמש גרפי, אמצעי אינטראקציה בין משתמש למחשב.
* **יוריסטיקות -** יוריסטיקות הן כללים פשוטים יחסית המשמשים מערכת הכרעת מצבים לקבלת החלטות. הן פועלות כמנגנונים קוגניטיביים זריזים המאפשרים לנו לקבל החלטות מהירות ויעילות תוך שימוש בכמות מוגבלת של מידע.
* **מערכת הכרעת מצבים –** היא מערכת קוגניטיבית המאפשרת קבלת החלטות, מורכבת מחישובים מבוססים יוריסטיקות. המערכת מזהה את המצב, אוספת מידע, מעריכה את האפשרויות שלה, בוחרת ומבצעת.

***תיאור הנושא -***

*תכולת המשחק:*

* לוח משחק משושה עם 61 שקעים.
* 28 כדורים, 14 מכל צבע (שחור ולבן).

*סידור לוח המשחק בהתחלה:*

* לכל שחקן 14 גולות בצבעים שחור או לבן, הגולות מסודרות בצורה הבאה, בשני קצוות הלוח, מסדרים את הגולות בשתי שורות מלאות, ובשורה השלישית במרכז את שלושתן הגולות הנותרות. בתמונה למטה ניתן לראות לוח בתחילת המשחק.



*מטרת המשחק:*

להיות השחקן הראשון שדוחף שש גולות של היריב אל מחוץ ללוח.

*מהלך השחקן:*

* כל שחקן בתורו מזיז קבוצה של כדורים מהכדורים שלו.
* ניתן להזיז כדור אחד או קבוצה של כדורים (עד שלושה כדורים) צמודים אחד לשני באותו הצבע לאורך קו ישר אופקית, אנכית או אלכסונית (דוגמא בתמונה משמאל).
* כדור או רצף של כדורים נעים עד שהם מגיעים לקצה הלוח, נתקלים בכדור או רצף של היריב.



*מהלכים בקו(In-Line Moves) –*

**בתורו, כל שחקן רשאי להזיז גולה בודדת או רצף של גולות משלהם צעד אחד. רצף מורכב משניים או שלושה גולות, מאותו הצבע, סמוכים אחד לשני בקו ישר.  
גולה או רצף של גולות, יכולים לזוז לכל כיוון בקו ישר עם הכיוון של הרצף, כל הגולות ברצף זזות צעד אחד באותו הכיוון, לתא ריק אלא אם מדובר במהלך SUMITO שיוסבר בהמשך (דוגמא למהלכים בקו ישר למעלה משמאל).**

*מהלכי צד(Broadside Moves) –*

**מהלכי צד, הם מהלכים בהם רצף הגולות זז צעד אחד לא בכיוון הרצף בקו ישר, במהלך צד כל הגולות חייבות לזוז לתא ריק ולא ניתן לדחוף גולות של היריב במהלך הזה (דוגמא למהלכי צד בתמונה למעלה מימין).**

*מהלך* ***SUMITO*** *(מהלך דחיפת גולות היריב):*

* שחקן יכול לדחוף את הגולות של היריב שנמצאים בחללים אחרים בעזרת דחיפת הגולות שלו בלבד בתזוזה בקו ישר.
* דחיפת גולות היריב תתאפשר אך ורק במצב בו מספר הגולות הדוחפות גדול ממספר הגולות הנדחפות (שלוש גולות יכולות לדחוף גולה אחת או שתיים ושתי גולות יכולות לדחוף גולה אחת בלבד של היריב).
* השחקן ידחוף את גולות היריב לחלל ריק או אל מחוץ ללוח.
* גולה הנדחפת אל מחוץ ללוח יוצאת מהמשחק לתמיד.
* גולה בודדה לא יכולה לדחוף גולה של היריב.
* אסור לדחוף גולות אויב הכרוכות בין גולות השחקן.
* בכל תור, לא ניתן להזיז יותר מ-3 גולות ידידותיות, ולכן לעולם לא ניתן לדחוף עמודה של שלוש של היריב. עמודה של 4 על 3 ומעלה אינה נחשבת **SUMITO**.



**מהלכים דחיפה חוקיים**

**מצבים בהם השחור לא יכול לדחוף את הלבן**

1. הגולה הלבנה כרוכה בין הגולות השחורות ואי אפשר לדחוף אותה.
2. הגולות השחורות והלבנות מופרדות בחלל ריק.
3. עמודה שזזה צעד רוחבי (לא בקו ישר) לא יכולה לדחוף אף פעם.

*סיום המשחק:*

* המשחק מסתיים או כשחקן אחד דוחף 6 כדורים של היריב מחוץ ללוח המשחק או שלא נותרו מהלכים חוקיים לשני השחקנים.

***רקע תיאורטי -***

*משחק אסטרטגיה:*

משחק אסטרטגיה הינו משחק השחקן צריך לפעול לפי תוכנית פעולה על מנת לנצח. תוכנית הפעולה תכלול מטרות ויעדים אותם השחקן רוצה להשיג במשחק ואת דרכי הפעולה למימושם. סביר להניח כי השחקן בעל תוכנית הפעולה הטובה יותר ינצח. ישנם סוגים רבים של משחקי אסטרטגיה כגון אסטרטגיה קבוצתית, אסטרטגיה מופשט, משחק מלחמה ועוד... המשחק Abalone משויך לסוג אסטרטגיה מופשט.

*משחק אסטרטגיה מופשט:*

משחק אסטרטגיה מופשט(Abstract strategy game) הוא משחק אסטרטגיה שבדרך כלל משוחק על גבי לוח וכולל את התכונות הבאות:

המשחק לא כולל מזל, השחקנים הם הגורמים המשפיעים היחידים על מצב המשחק(מלבד מי מתחיל).

שני השחקנים יודעים הכל על מצב המשחק.

המהלכים של השחקנים מתבצעים בזה אחר זה בסדר.

מכאן שמשחק אסטרטגיה מופשט הוא משחק שנקבע לפי החשיבה והאסטרטגיות של השחקנים עצמם ולכן זהו משחק בו שחקנים צריכים להתפתח, להתאמן ,להמציא ולגלות תוכניות פעולה רבות על מנת להשתפר.

*הכרעת מצבים:*

במשחקי אסטרטגיה מופשט כגון אבלון, שחמט, דמקה מספר המצבים של המשחק הוא עצום ולכן לא תמיד ניתן לסרוק את המשחק עד הסוף ולהבין מה המהלך האופטימאלי, לכן נעזר בשיטת הכרעת מצבים. לפי שיטה זו נקבע למשחק בו אנו משחקים כמה פרמטרים שיבחינו אם מצבנו במשחק יותר טוב או פחות

טוב מזו של היריב. בעזרת פרמטרים אלו נבחן את אופציות המהלכים שיש לנו ונבין מהו המהלך האופטימאלי. כמובן שישנו סיכוי כי מהלך זה יתברר לא כמהלך האופטימאלי עקב רצף אירועים שעלול להתרחש בעתיד המשחק כיוון שלא בדקנו מה היריב יכול לעשות בתגובה אך זוהי דרך יעילה וטובה מאוד להגיע למהלך שהוא אחד מהטובים שישנו.

***תיאור הבעיה האלגוריתמית –***

האתגר המרכזי הוא יצירת אלגוריתם AI חכם שמסוגל לשחק אבאלון ברמה גבוהה. האלגוריתם צריך לקבל החלטות אסטרטגיות מורכבות תוך ניהול יעיל של הגולות שלו והתחשבות במצבי המשחק המשתנים.  
  
*הבעיות המרכזיות:*

* **הכרעת מצבים:** הערכת מצב הלוח בכל תור, תוך התחשבות במיקום גולות היריב, מיקום הגולות שלך, המרחק ביניהם, וחוזק וחולשות היריב.
* **ניהול יעיל של גולות:** ניהול הגולות לא רק לפי מספרן אלא גם לפי מיקומן על הלוח ומקבצן.
* **חיזוי תנועות היריב:** ניסיון להבין את תנועות היריב הצפויות ותכנון מהלכים בהתאם.

*פתרון אפשרי:*

שימוש במערכת הכרעת מצבים מתקדמת שתאפשר לאלגוריתם:

* **להעריך את מצב הלוח:** ניתוח מיקום הגולות, מרחקים, חסימות, מקבצים, גולות בסכנה ועוד.
* **חיזוי תנועות היריב:** לנסות להבין מה המהלך הבא שהשחקן יבצע ולפי זה לבצע את שלנו.
* **להעריך את חוזק וחולשות היריב:** זיהוי נקודות תורפה אצל היריב ובהתאם לתכנן מהלכים מותאמים.
* **לבחור אסטרטגיה אופטימלית:** בחירה בין מהלכים התקפיים או הגנתיים.
* **לנהל את הגולות בצורה יעילה:** ניצול מיקום הגולות מיטבי, יצירת מקבצי הגנה והתקפה חזקים.

*תהליכים עיקריים בפרויקט:*

* תכנון מבנה המשחק, ארכיטקטורת התוכנה, מבנה הלוח, הכלים וכו'
* פיתוח לוגיקת המשחק, כללי תנועה, תקיפה, ניצחון וכו'
* פיתוח ממשק מינימלי, יצירת גרפיקה ממשק משתמש מינימלי לאחד על אחד.
* פיתוח אלגוריתם לקבלת החלטות במשחק והטמעתו לשחקן מול המחשב.
* שדרוג הממשק לרמת גמר ועיצוב ממשק משתמש נוח וידידותי.
* בדיקות אינטגרציה במערכת, שדרוגים נחוצים ותיקון באגים.

***סקירת אלגוריתמים בתחום הבעיה –***

**Monte Carlo Tree search** **–**

אלגוריתם חזק שמתמודד טוב עם מרחבי מצבים גדולים בעזרת סימולציות רנדומליות של משחקים. אופציה טובה למשחקים שבהם עומק גדול של עץ חיפוש אינו פרקטי, ודיוק החיזוי תלוי בכמות הסימולציות.

**Reinforcement Learning (RL) –**

זו שיטה בה סוכן לומד איך לנהוג בסביבה על ידי ניסוי וטעיה, תוך מקסום פונקציית פרס. הייתרון של האלגוריתם הזה זה, היכולת שלו להתמודד עם מצבים חדשים וללמוד מתוך ניסיון, ללא צורך בהגדרה מפורשת של כל המצבים. החיסרון שלו זה שתהליך הלמידה יכול להיות ארוך ודורש משאבים רבים.

**Deep Reinforcement Learning (DRL) -**

שילוב של neural networks עם RL, מאפשר לסוכנים ללמוד מתוך נתוני קלט מורכבים ולפתח אסטרטגיות משחק מתקדמות. הייתרון של גישה זו זה היכולת לזהות תבניות ואסטרטגיות מורכבות בלי שצריך להגדיר כללים מוגדרים מראש.

**הערכת מצב הלוח מבוססת היוריסטיקה –**

פונקציות הערכת מצב הלוח בעזרת היורסטיקות הן טכניקה כללית להשתמש בה במגוון רחב של משחקים. האלגוריתם מעריך את מצב הלוח במשחק אבאלון על ידי שימוש בכליים היוריסטים. היתרונות של אלגוריתם זה הם, שהוא פשוט וקל ליישום, מהיר ויעיל. החסרונות הם שההערכה אינה תמיד מדויקת, מכיוון שהיא מבוססת על כללים היוריסטים ולא על ניתוח מעמיק של המשחק, על מנת לפתח היורסטיקות טובות דרוש ידע אנושי נרחב במשחק ובאסטרטגיות שלו.

***אלגוריתם נבחר –***

**אלגוריתם הערכת מצב לוח מבוסס היוריסטיקה:**

**תיאור האסטרטגיה/האלגוריתם:**

1. **קבלת קלט:**  מצב הלוח, הכולל את לוח המשחק עם מיקומי הגולות של השחקנים וניקוד.
2. **חישוב ניקוד:** על לפי מספר היורסטיקות(אסטרטגיות) ניתן ציון למצב הלוח, דוגמאות לאותן היורסטיקות הן, על פי מרחק הגולות מהמרכז, מקבצן, תיעדוף קבוצות של עד שלוש, דחיפה אל מחוץ ללוח ועוד.
3. **החזרת פלט:** החזרת ציון הסוכם את הציונים מכל ההיורסטיקות, ציון זה מייצג את שווי מצב הלוח הספציפי הזה עבור שחקן מסוים.

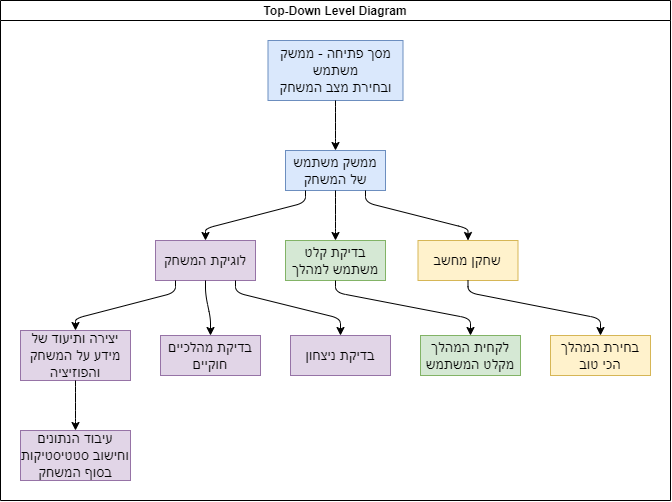
**שימוש במשחק שלי:**

כל פעם שיגיע תורו של שחקן המחשב, נחשב את המהלכים האפשריים של אותו השחקן, לאחר מכן נעבור על כל המהלכים, נבצע את המהלך ונקבל מצב לוח חדש, את המצב הזה נשלח לאלגוריתם הכרעת המצבים שלנו, ונקבל את הציון של אותו המהלך. ועכשיו נעשה UNDO למהלך שביצענו ונעבור למהלך הבא. בממוצע יש באזור השישים מהלכים אפשרים בכל תור (משתנה בהתאם להתקדמות המשחק). על מנת לשפר את היעילות נכין מפת גיבוב שהמפתח שלה זה קוד האש של מצב לוח והערך הוא רשימה של המהלכים הכי טובים לאותו מצב לוח. וככה במקרים שחישבנו כבר את המהלכים הכי טובים למצב לוח מסוים שחזר על עצמו לא נצטרך לחשב שוב.

**נימוק לבחירה שלי:**

* **פשוט וקל ליישום:** ניתן ליישם בקלות את האלגוריתם במשחק אבאלון בעזרת java.
* **מהיר:** זמן חישוב קצר יחסית.
* **יעיל:** יכול להעריך בצורה טובה את מצב המשחק (במידה ופונקציות ההיוריסטיקות טובות ומדויקות).
* **גמישות:** ניתן להתאים את ההיוריסטיקות בהתאם לסגנון משחק ספציפי.
* **שקיפות:** קל להבין את אופן פעולת האלגוריתם.
* **יכולת שיפור:** ניתן לשפר את דיוק ההערכה על ידי פיתוח היוריסטיות מתקדמות יותר.

***Top-down Level Design -***



# ***תיאור מילולי של ארכיטקטורת הפרויקט -***

* ***מסך******פתיחה*** *-*
* בחירת שחקנים(מחשב או אדם).
* בחירת השחקן המתחיל.
* מקבל קלט מהמשתמש עד לחיצת הכפתור START.
* ***ממשק המשתמש של השחקן*** *–*
* מקבל קלט מהמשתמש (אם שחקן אדם).
* מקבל פלט ממחלקת ה – AI (אם שחקן מחשב).
  + ***לוגיקת המשחק –***

מקבלת קלט שהוא פוזיציית התחלת המשחק, מקבלת קלט ממשק המשתמש.

* + - ***בדיקת ניצחון*** –

מקבלת קלט מהלוגיקה ומחזירה אם שחקן ניצח או לא או אם התקיים תיקו.

* + - ***בדיקת המהלכים –***

מקבלת קלט מהלוגיקה מחשבת ומחזירה את המהלכים החוקיים.

* + - ***יצירת ושימור מידע לגבי המשחק –***

בסוף המשחק שומר תיעוד של המשחק וסטטיסטיקות ונתונים עליו בקובץ ומציע לשחקן לראות ולהוריד את הקובץ.

* + ***שחקן מחשב –***

מחשב את המהלך הטוב ביותר עבור השחקן שלו.

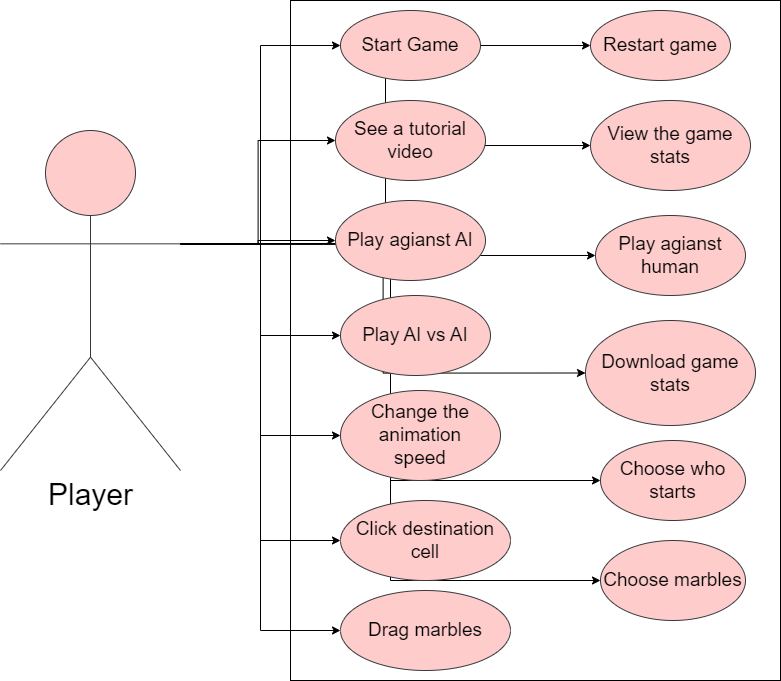
* + ***בדיקת הקלט של המשתמש –***

בודק את הקלט של המשתמש, האם המהלך תקין או לא פולטת ללוגיקה את המהלך אם הוא חוקי.

* + - ***עדכון הגרפיקה –***

עדכון הגרפיקה לפי המהלך שהתבצע על ידי האדם או המחשב ואם המשחק נגמר או לא.

***UML Use cases -***

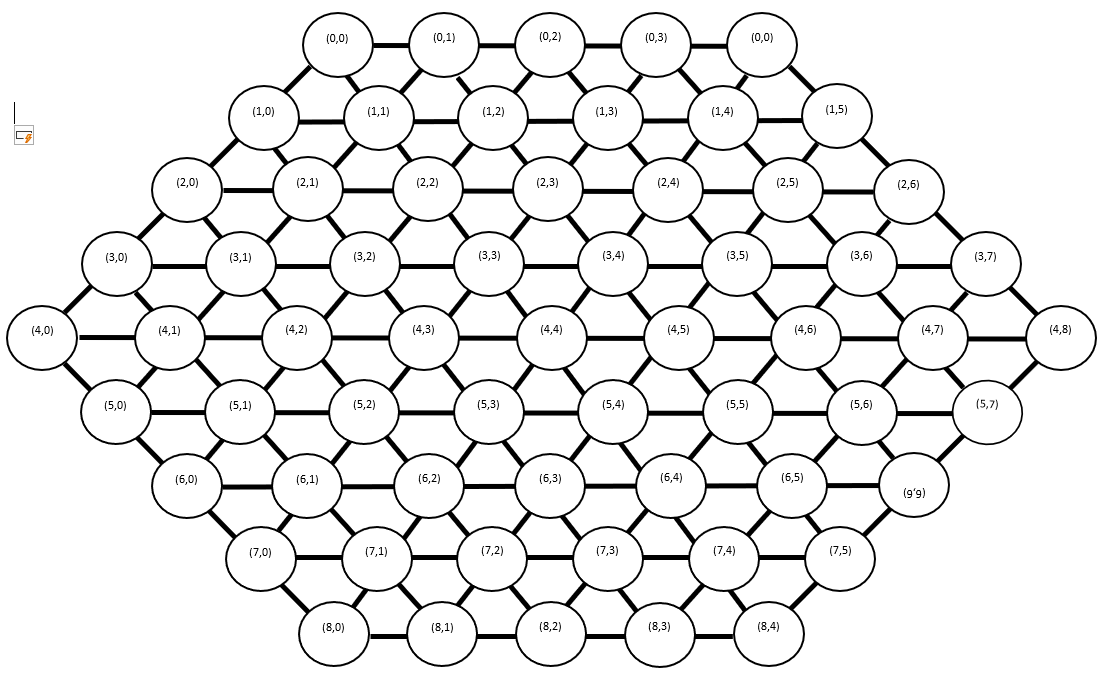


***מבנה נתונים -***

**Map<Cell ,Direction> -** השתמשתי במפה כדי שכל אובייקט של תא בלוח ישמור יחזיק הופעה של כל האובייקטים של התאים השכנים שלו לפי כיוונם, הכיוון הוא מסוג ENUM. מאפשר לי מציאה יעילה של תאים שכנים ולהבנה של היחסים המרחביים שלהם, שזה הבסיס לתנועה במשחק אבאלון. הבחירה בCell כמפתח ו Direction כערך מאפשרת חיפוש מהיר של כיווני תאים שכנים ביחס לתא הנוכחי.

**Hashset<> -** השתמשתי בהאש סט כדי להחזיר אוסף של תאים שכנים. אוסף זה נבחר בשל התכונות שלו של ייחודיות וזמני חיפוש יעילים, החשובים בעת ניהול קבוצת תאים סמוכים ללא כפילויות. הוא מאפשר לבצע עליו בדיקות נוכחות של שכן ספציפי במהירות.

**Map<Cell, Map<Cell, Direction>> -** מבנה הנתונים הזה הוא מבנה הנתונים המרכזי המייצג את לוח המשחק שלי. הוא מקשר כל תא Cell ל Map אחר, אשר מקשר עוד אובייקטי Cell שכנים לכיוון המתאים שלהם ביחד לתא המפתח הנוכחי. בעצם מבנה זה מייצג מפת סמיכויות של גרף קשיר בלתי מכוון. מבנה הנתונים הזה מתאים במיוחד למשחקי עם לוח בצורת משושה שבהם לכל תא יכולים להיות עד שישה שכנים, כל אחד בכיוון שונה. הוא מאפשר חיפוש יעיל של שכני התא והכיוון של כל שכן, דבר שהוא קריטי ליישום לוגיקת משחק שתלויה ביחסים מרחביים, כמו תנועה. מצורף למטה תמונה להמחשה של הלוח מייוצג על ידי מבנה הנתונים.



**int[][] –** מערכים דו ממדיים של מספרים שלמים משמשים לייצוג הכיוונים ביחס לתא על הלוח. שלושה מערכים כאלה אחד לכל חלק של הלוח, תחתון עליון ומרכז כדי לתת חשבון לתכונות גיאומטריות ייחודיות של רשת משושה בייצוג מהסוג שבחרתי. מערכים אלו מקודדים את ההיסט בתא כדי למצוא תאים שכנים. גישה זו מפשטת את החישוב של שכנים על ידי מתן דרך ישירה לעבור על כיוונים אפשריים ולמצוא תאים סמוכים בהתבסס על מיקומם.

**HashMap<> -** נשתמש במפת גיבוב בפונקציות שמאתחלות את הלוח ויצירת סמיכויות בין תאים. מבחינת יעילות מפת גיבוב הכי טובה במקרה הזה כי הסיבוכיות היא זמן קבוע, דבר שהוא חשוב מאוד לניהול מצב המשחק שבו נדרשת גישה תכופה ועדכונים לתאים ולשכנים שלהם.

**Collection<Cell> -** בפונקציה getCells מחזירים אוסף של תאים, המספקת דרך כללית לגישה לכל התאים על הלוח מבלי הצורך לחשוף ולהעביר את מבנה הנתונים הבסיסי שמחזיק את הלוח. זה מאפשר קוד גנרי וגמיש יותר מכיוון שאפשר לשנות את סוג האוסף הבסיסי מבלי לשנות את הממשקים בהם משתמש שאר התוכנית. זה שימושי בכל הפעולות שצריכות לעבור על התאים, כמו אתחול לוח המשחק ובדיקת המהלכים.

**List<Cell> -** מסוג ArrayList משמש לאחסון אוסף של אובייקטי Cell שמייצגים את התאים בלוח ואיזה גולה נמצאת עליהם אם נמצאת. נשתמש ברשימה במקרים האלה בגלל שגודלן דימני והקלות שלהן בסינון וגישה לרכיבים. זה מאוד נוח במהלכים הכוללים מספר גולות, וכאשר הסדר יכול להיות משמעותי לקביעת תקינות המהלך ולביצועו בסדר מסוים.

**Map<Cell, Integer> -** מסוג HashMap משמש למעקב אחר מצב התאים שהשתנו כתוצאה מהמהלך, ומאפשר את פונקציית הביטול. המפתחות הם אובייקטי Cell המושפעים מהמהלך, והערכים הם המצבים שלהם לפני ביצוע המהלך. המפה הזאת חיונית לשחזור יעיל של הלוח למצב הקודם שלו, פונקציה חיונית עבור מערכת הכרעת המצבים שמדמה מהלכים מעריכה ומחזירה למצב המקורי.

**Comparator<Cell> -** בפונקציה SortMarbles משתמשים בקומפרטור כדי להגדיר סדר מיון מותאם אישית של אובייקטי Cell . אני מגדיר את זה אישית כדי לסדר את הגולות לפי הקואורדינטות שלהן, ולהבטיח שפעולות הדורשות גולות מסודרות יעבדו טוב.

**List<Cell> -** נוסף כדי לשמור את גולות השחקן וגולות היריב לAI, עוד שתי רשימות לשמירת יעדים פוטנציאליים לזוז אליהם, ועוד רשימה זמנית לשמירת המהלכים האפשריים בפונקציה שמחשבת אותם. בחרתי להשתמש בממשק List היא בשל יכולות האוסף המסודר שלו, המאפשר גישה לפי אינדקס לאלמנטים שלו. שימושי במיוחד בלולאות שחוזרות על גולות ותאים כדי ליצור מהלכים או לעדכן את לוח המשחק, בנוסף הדינמיות של הגודל הכרחית.

**ArrayList<Move> -** למעקב אחרי המהלכים האפשריים של שני השחקנים, והמהלכים הכי טובים שנקבעו על ידי מערכת הכרעת המצבים. בחרתי בArrayList כי היא מספקת גישה ואחסון יעילים לרשימות המהלכים, בחרתי בה בגלל זמני האיטרציה והאיתור המהירים שלה, דבר שחיוני עבור מערכת הכרעת המצבים שלי כדי שהיא תנתח ותשווה במהירות בין המהלכים הפוטנציאלים השונים.

**Set<Move> - לאחסון זמני של מהלכים ייחודיים בפונקציה שמייצרת את המהלכים האפשריים, לפני שהיא מעבירה אותם לArrayList. בחרתי להשתמש בSet כי הוא מאפשר לשמור אלמנטים ייחודיים, ומבטיח שלא ייחשבו מהלכים כפולים. זה שימושי כשיוצרים מהלכים כדי להימנע מעיבוד של אותו המהלך כמה פעמים.**

**List<Cell> - לאחסון גולות השחקן ולשמירה על היסטוריית המהלכים לטווח קצר על מנת לזהות לולאות במשחק.**

**Stack<Move> - שימוש למעקב אחרי שתי המהלכים האחרונים. השתמשתי במחסנית בגלל התכונה LIFO, מה שופך אותה לאידאלית לפונקציונליות של ביטול.**

**מצורף בנספחים תמונות עם סרטוטים שקשורים למבנה הנתונים הראשי של הלוח ותזוזות שלו.**

***תיאור סביבת העבודה ושפת התכנות –***



שפת תכנות: **JAVA 18.0.2.1**

סביבת העבודה:**Code Microsoft Visual Studio**  
 **Version 1.84.2**

***אלגוריתם ראשי (פסיאדו קוד) -***

**Get All Potential Moves for Both Players**

* For the computer player and the opponent, calculate all possible moves given the current board state.
* Filter potential destination cells based on their proximity to the player's and opponent's marbles.
* Update lists of marbles for both players.

**Computer's Turn: Select the Best Move**

* Evaluate each potential move by simulating its outcome and scoring the resulting board state.
* Compare the scores to identify the best move(s).
* If multiple best moves are found, select one at random.
* Update the game board to reflect the chosen move.

**Evaluate Board State for a Move**

* Simulate a move and analyze the new board state.
* Consider factors like the position of marbles, potential for marbles to be pushed off, formation of marble groups, and proximity to the center.
* Calculate a score based on these factors to assess the move's effectiveness.

**Heuristics for Scoring**

* Gravity Center: Favor positions closer to the center of the board.
* Pushed Off: Reward moves that push opponent marbles off the board.
* Keep Packed: Encourage keeping marbles close to each other.
* Marbles in Danger: Penalize positions that leave marbles vulnerable to being pushed off.
* Evaluate Group Scores: Score based on the formation of marble groups in various directions.

**Move Execution and Reversal**

* For move evaluation, simulate the execution of a move by applying it to the board.
* After evaluation, reverse the move to restore the board to its original state before proceeding to the next move.

***תיאור API -***

את ממשק המשתמש הגרפי (GUI) של האפליקציה בניתי עם JavaFX, הממשק מספק חווית משתמש נוחה יפה ואינטראקטיבית ויזואלית למשחק האבאלון.

**JavaFX –** כל ממשק המשתמש בנוי על JavaFX , שמאפשרת ליצור אפליקציות במחשב עם ממשקים גרפיים. JavaFX מספקת רכיבים כגון כפתורים, תוויות ותמונות, בתוך קונטיינרים כמו HBox, VBox וכדומה, כדי לסדר את האלמנטים של ממשק המשתמש בצורה מסודרת.

**FXML –** הFXML של JavaFX משמש להגדרת הפריסה של הממשק הגרפי. נגדיר בו את המבנה של ממשק המשתמש, כולל מיקום של כפתורים, תוויות ועוד אלמנטים. נגדיר בו מטפלי אירועים שמחוברים לאלמנטים אלה של ממשק המשתמש, ומחברים אותם ללוגיקה של האפליקציה.

**CSS –** משמש לעיצוב ממשק המשתמש, התאמה אישית של רכיבי ממש המשתמש, כולל צבעים, גדלים וגופנים.

**MediaView - MediaPlayer –** מחלקות MediaView-MediaPlayer של JavaFX משמשות לשילוב ושליטה בהפעלת וידאו בתוך האפליקציה. תכונה זו משמשת להצגת מדריך וידאו "איך לשחק אבאלון" מתפריט האפליקציה, ומציעה למשתמשים סרטון הדרכה על חוקי המשחק.

**אנימציה –**  האנימציה של JavaFX, כמו FadeTransition, TranslateTransition וTimeline משמשות ליצירת אפקטים חלקים לממשק המשתמש של המשחק. זה כולל הנפשות לתנועות של כדורים, סימון תורות שחקן וסימון אם מהלך הוא חוקי או לא.

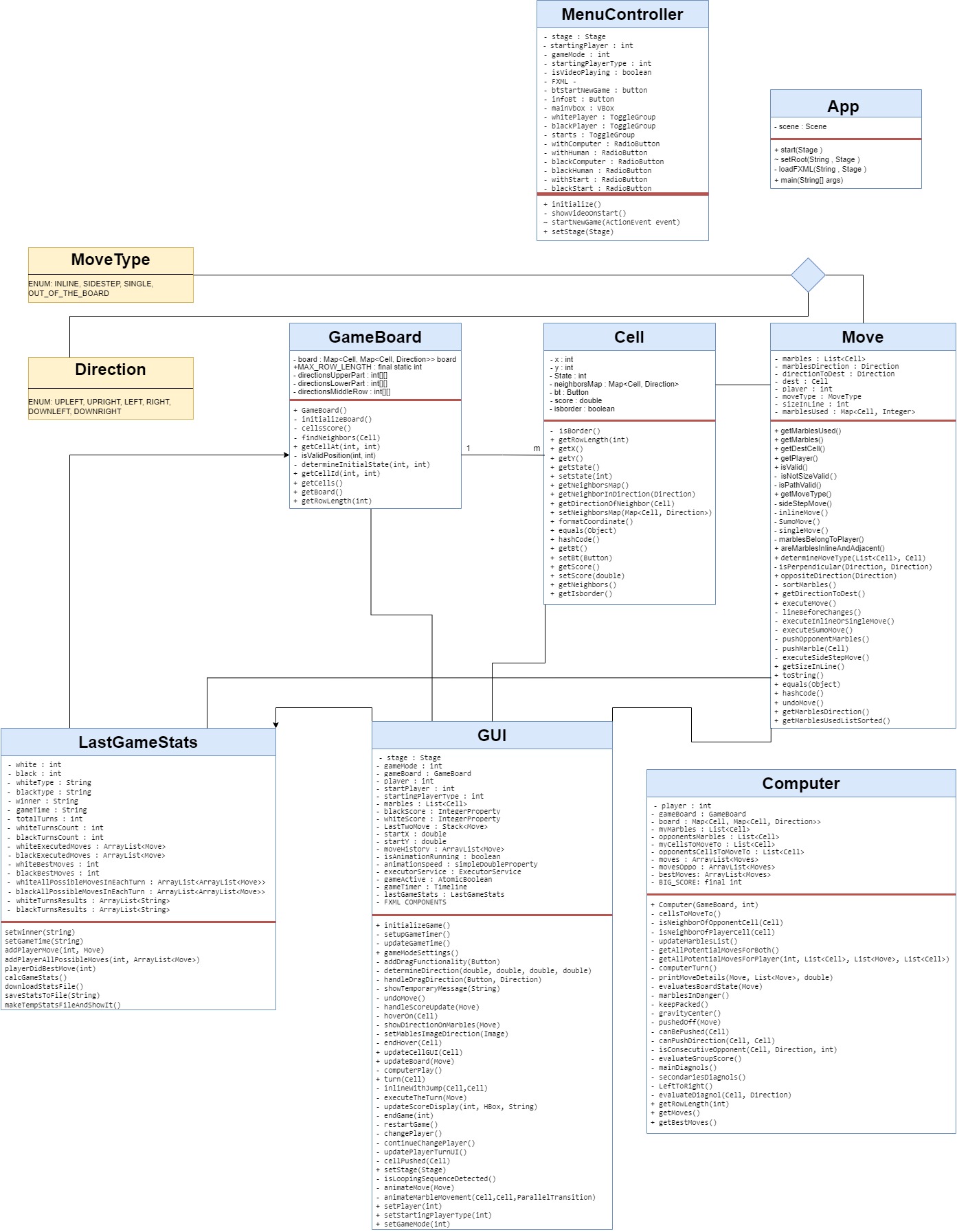
במסך הראשוני נבחר את הגדרות המשחק ובלחיצה על כפתור START נעבור לחלון המשחק ונתחיל לשחק.

על מנת להזיז גולות במהלך התור צריך לבחור את הגולות אותן רוצים להזיז על ידי לחיצה על עד שלוש גולות חוקיות או על ידי סימון שתי גולות והגולה שביניהן תבחר גם. לאחר מכן נצטרך לבחור לאן נרצה להזיז את הגולות, על ידי ריחוף סמן העכבר מעל התאים הריקים או גולות אויב סמוכים ניתן יהיה לראות אם אפשר לזוז לשם על ידי חצים שיופיעו על הגולות של השחקן ובלחיצה המהלך יבוצע והגולות יזוזו. דרך אחרת לבחור את היעד של הגולות היא על ידי גרירה לכיוון מסוים, ואם המהלך בכיוון הזה חוקי המהלך יבוצע והתור יעבור לשחקן השני.

את הניקוד ניתן לראות בחלקו העליון של המסך ועל ידי מספר ועל ידי תמונות של הכדורים שנדחפו.

בצדו השמאלי של המסך ישנו בר, שניתן להעלות ולהוריד והוא שולט על מהירות האנימציה במידה והשחקן רוצה להאיץ או להאט את מהירות האנימציה.

***UML Class Diagram -***



***הפונקציות/המחלקות הראשיות בפרויקט –***

## **שם מחלקה: GUI**

**מחלקה:** בקרת המשחק

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **פונקציה** | **פרמטרים** | **מחזירה** | **יעילות** | **תיאור** |
| restartGame | ללא פרמטרים | ללא החזרה | יכול להיות לא מאוד יעיל בגלל טעינה של סצנת התפריט | מפעיל מחדש את המשחק על ידי טעינת סצנת תפריט המשחק |
| changePlayer | ללא פרמטרים | ללא החזרה | יעיל | מחליף בין שחקנים ומעדכן את מחוון התור החזותי. |
| isLoopingSequenceDetected | רשימה של מהלכים | Boolean | לינארי בגודל היסטוריית המהלכים במקרה הגרוע | בודק את היסטוריית המהלכים עבור רצפים חוזרים שאומרים שהמשחק נתקע בלולאה |

**מחלקה:** אנימציה

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **פונקציה** | **פרמטרים** | **מחזירה** | **יעילות** | **תיאור** |
| animateMove | Move | ללא החזרה | תלוי במספר הגולות במהלך | מטפל באנימציה של תנועת גולות לפי סוג המהלך |
| animateMarbleMovement | startCell, endCell | ללא החזרה | לינארי | אנימציה של גולה בודדת שזזה מתא התחלה לתא סיום |

**מחלקה:** עדכון ממשק משתמש

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **פונקציה** | **פרמטרים** | **מחזירה** | **יעילות** | **תיאור** |
| updatePlayerTurnUI | ללא פרמטרים | ללא החזרה | זמן קבוע | מעדכן את התצוגה של התור של השחקן הנוכחי |
| updateCellGUI | Cell | ללא החזרה | זמן קבוע | מעדכן את הייצוג החזותי של תא על הלוח לפי מצבו |
| cellPushed | Cell | ללא החזרה | זמן קבוע | מחליף כדור בכדור מעומעם כשכדור נבחר |

**מחלקה:** ניהול מצב המשחק

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **פונקציה** | **פרמטרים** | **מחזירה** | **יעילות** | **תיאור** |
| setPlayer | Player(int) | ללא החזרה | זמן קבוע | מגדיר את השחקן הנוכחי בתחילת המשחק |
| setStartingPlayerType | startingPlayerType  (int) | ללא החזרה | זמן קבוע | מגדיר את סוג השחקן ההתחלתי |
| setGameMode | gameMode(int) | ללא החזרה | זמן קבוע | מגדיר את מצב המשחק (אדם מול מחשב או מחשב מול מחשב או אדם מול אדם) |

**פסיאודו קוד -**

**restartGame:**

1. נסה להעלות את סצנת התפריט (menu.fxml)
2. סגור את הסטייג של המשחק הנוכחי
3. תפוס כל שגיאה במהלך הטעינה והצג הודעת שגיאה

**changePlayer:**

1. Player שווה לשחקן שלא שיחק עכשיו
2. יצירת ציר זמן שבודק כל 100 מילי שניות אם אין אנימציה פעילה כדי לעדכן שחקן וממשק משתמש

**2.1** אם אין, קרא ל continueChangePlayer

**isLoopingSequenceDetected:**

1.אם גודל היסטוריית המהלכים קטן מעשר

1.1 החזר שקר

2. לכל גודל רצף מ 2 עד חצי מגודל ההיסטוריה

2.1. בדוק אם הרצף חוזר על עצמו בMoveHistory

2.1.1. אם נמצאה לולאה החזר אמת

2.2. אם לא נמצאה לולאה החזר שקר

**animateMove:**

**1.** יצירת ParallrlTramsition להנפשת התנועות בו זמנית

2. isAnimationRunning = true

3. **אם המהלך הוא מהלך צד**

**3.1. כיוון = כיוון המהלך**

**3.2. לכל תא במהלך**

**3.2.1. תא יעד = תא השכן בכיוון של המהלך**

**3.2.2**.( **תא, תא יעד, מעבר**animateMarbleMovement(

אחרת:

4. move.undoMove()

5. **כיוון = כיוון המהלך**

**6.** כיוון הפוך = כיוון הפוך של כיוון

7. destPlayerNeighbor = null

8. אם כיוון שונה מNull

8.1. destPlayerNeighbor שווה לכיוון ההפוך מכיוון המהלך

9. זיהוי תאים המעורבים בדחיפה (תא יעד, תאים פוטנציאליים נדחפים)

10.יצירת רשימה לאחסון תאים המעורבים בהנפשה, marbles = רשימה חדשה

11. הוספת תא היעד לרשימה אם אינו ריק

12. הוספת גולות יריבות נדחפות, במידה וקיימות, לרשימה

13. זיהוי גולת יריב נדחפת שבאה בכיוון ההפוך

14. ביצוע המהלך בפועל על מצב הלוח, move.executeMove

15. הנפשת התנועה לכל תא המעורב במהלך, אם גודל רשימת הגולות הוא 2 אז:

15.1. animateMarbleMovement(marbles.get(1), move.getDestCell(),ParallrlTramsition )

15.2. **אחרת,** marbles **בטווח 1 עד אורך i**

**15.2.1.** startCell = marbles.get(i)

**15.2.2.** endCell = cell

**15.2.3.** animateMarbleMovement(startCell , endCell ,ParallrlTramsition )

16. **עבור כל תא ברשימה תעדכן את התמונה הסופית שלו**

**17. בסוף האנימציה** isAnimationRunning **= שקר**

**updatePlayerTurnUI:**

עדכן תצוגת תור השחקן:

בהתאם לצבע השחקן (לבן או שחור), עדכן טקסט, צבע טקסט, וצבע רקע.

**updateCellGUI:**

1. קבע תמונה בהתבסס על מצב התא

2. צור תצוגת תמונה מהתמונה

3. עדכן את התצוגה הגרפית של התא עם התמונה החדשה

**cellPushed:**

1. קבע תמונה בהתבסס על מצב הלוח

2. צור תצוגת תמונה מהתמונה

3. הבהר מעט את תצוגת התמונה כדי לסמן שהתא נלחץ

4. עדכן את התצוגה הגרפית של התא עם התמונה החדשה

**setPlayer:**

1. קבל int המייצג את השחקן הנוכחי

2. player = לשחקן הנוכחי

3. startPlayer = שחקן ההתחלה שמתחיל את המשחק

**setStartingPlayerType:**

1. עדכן את המשתנה startingPlayerType בהתאם לסוג השחקן ההתחלתי שהתקבל

**setGameMode:**

1. עדכן את המשתנה gameMode בהתאם למצב המשחק שהתקבל

## **שם מחלקה: Gameboard**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **פונקציה** | **פרמטרים** | **מחזירה** | **יעילות** | **תיאור** |
| Gameboard | ללא פרמטרים | GameBoard | O(1) | בנאי שמאתחל לוח משחק ריק |
| initializeBoard | ללא פרמטרים | ללא החזרה | O(n^2) | מאתחל את הלוח על ידי יצירת תאים, הקצאת מצבים התחלתיים שלהם והקמת קשרי סמיכות ביניהם |
| findNeighbors | Cell | Map<Cell, Direction> | O(n) | מוצא ומחזיר מפה המכילה את כל התאים השכנים התקפים של תא נתון, יחד עם הכיוון שבו הם נמצאים |
| getCellAt | Int x, int y | Cell | O(n) | מחזיר את אובייקט ה Cell בקואורדינטות שצוינו. |
| isValidPosition | Int x, int y | Boolean | O(1) | קובע אם מיקום נתון קיים בלוח המשחק המשושה |
| determineInitialState | Int x, int y | Int | O(1) | קובע את מצב ההתחלה של כל תא בהתבסס על מיקומו |
| getCellId | Int x, int y | String | O(1) | יוצר מזהה ייחודי עבור תא נתון |
| getCells | ללא פרמטרים | Collection<Cell> | O(n) | מחזיר אוסף של כל התאים בלוח |
| getBoard | ללא פרמטרים | Map<Cell, Map<Cell, Direction>> | O(n) | מחזיר את לוח המשחק המלא, כולל מידע על קשרי סמיכות בין תאים |
| getRowLength | Int x | Int | O(1) | מחזיר את אורך השורה הנתונה בלוח |

**פסיאודו קוד -**

**GameBoard:**

1. צור מפה חדשה לאחסון תאים (לוח המשחק).

2. עבור על כל אפשרויות המיקום על הלוח:

2.1. בדוק אם המיקום תקף בלוח המשושה.

2.2. אם כן, קבע מצב התחלה עבור התא (שחור, לבן או ריק).

2.3. צור אובייקט תא חדש עם קואורדינטות ומצב התחלה.

2.4. אתחל מפת קשרי סמיכות עבור התא.

2.5. הוסף את התא והקשרי סמיכות שלו למפת הלוח.

3. קבע ציונים לתאים.

**findNeighbors:**

1. צור מפה חדשה לאחסון תאים שכנים.

2. קבל את קואורדינטות התא.

3. קבע את מערך הכיוונים המתאים בהתבסס על השורה של התא.

4. עבור על כל אפשרויות הכיוונים:

4.1. חשב את הקואורדינטות של התא השכן.

4.2. קבל את אובייקט התא השכן.

4.3. אם קיים תא שכנים תקף, הוסף אותו למפה עם הכיוון המתאים.

5. החזר את מפת התאים השכנים.

**getCellAt:**

1. עבור על כל התאים בלוח.

2. אם קואורדינטות התא תואמות את הקואורדינטות הנתונות, החזר את התא.

3. אם לא נמצא תא מתאים, החזר null

**isValidPosition:**

1. בדוק את ערך x

1.1. אם x הוא 0 או 8, בדוק אם y בין 0 ל4.

1.2. אם x הוא 1 או 7, בדוק אם y בין 0 ל5.

1.3. אחרת, בדוק אם y בין 0 ל8

2. אם התנאי מתקיים, המיקום תקף.

3. אחרת, המיקום לא תקף.

**determineInitialState:**

1. בדוק את ערך x

1.1. אם x הוא שורה ראשונה או שורות 2,3 וקואורדינטת y בין 2 ל4, קבע מצב 2 (שחור).

1.2. אם x הוא שורה אחרונה או שורות 2,3 וקואורדינטת y בין 2 ל4, קבע מצב 1 (לבן).

2. אחרת, קבע מצב 0 (ריק).

**getCellId:**

1. צור מחרוזת המכילה את הקידומת "bt" והוסף למחרוזת את ערך x וערך ה y מופרדים במקף.

2. החזר את המחרוזת שנוצרה

**getRowLength:**

1. בדוק את ערך x

1.1. אם קטן מ4 החזר x + 5

1.2. אן x שווה ל4 החזר את אורך השורה האמצעית.

1.3. אחרת, החזר x-13

## **שם מחלקה: Cell**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **פונקציה** | **פרמטרים** | **מחזירה** | **יעילות** | **תיאור** |
| Cell | Int x, int y, int state | ללא החזרה | O(1) | בנאי שמאתחל תא חדש |
| getNeighborsMap | ללא פרמטרים | ללא החזרה | O(1) | מחזירה מפה של תאי השכן של התא עם הכיוונים אליהם |
| getNeighborInDirection | Direction | Cell | O(n) | מחזירה את תא השכן בכיוון נתון |
| isBorder | ללא פרמטרים | Boolean | O(1) | קובעת האם התא הוא תא שפה |

**פסיאודו קוד -**

**getNeighborInDirection:**

1. בדוק אם מפת השכנים לא קיימת

1.1. אם השכנים לא קיימת החזר NULL

2. עבור על כל תא שכנים במפת השכנים

3. בדוק אם הכיוון של התא השכן תואם את הכיוון המבוקש

3.1. אם הכיוונים תואמים, החזר את תא השכן הנוכחי

4. אם לא נמצא תא שכנים בכיוון המבוקש, החזר NULL

**isBorder**:

1. בדוק את ערך ה x של התא

1.1. אם ה x שווה ל0 או 8 (שורות ראשונה ואחרונה), התא הוא תא שפה

1.2. אחרת, עבור לשלב הבא

2. בדוק את ערך ה y של התא:

2.1. אם ה y שווה ל0 (שורה ראשונה), התא הוא תא שפה

2.2. אחרת, עבור לשלב הבא  
3. חשב את אורך השורה בהתבסס על ערךx

3.1. אם אורך השורה שווה לy + 1, התא הוא תא שפה בקצה השורה

3.2. אחרת, התא אינו תא שפה

## **שם מחלקה: Move**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **פונקציה** | **פרמטרים** | **מחזירה** | **יעילות** | **תיאור** |
| isValid | ללא פרמטרים | Boolean | O(n) | בודקת האם המהלך תקין |
| determineMoveType | List,Cell | MoveType | O(1) | קובעת את סוג המהלך |
| sortMarbles | ללא פרמטרים | ללא החזרה | O(nLog(n)) | מסדרת את רשימת הגולות לפי מיקומן על הלוח |
| executeMove | ללא פרמטרים | ללא החזרה | O(n) | מבצעת את המהלך על הלוח |
| undoMove | ללא פרמטרים | ללא החזרה | O(n) | מבטלת את המהלך שבוצע |

**פסיאודו קוד -**

**isValid:**

1. מיון גולות, מיון רשימת הגולות לפי מיקומן על הלוח בעזרת הפונקציה sortMarbles

2. בדיקת גודל הקבוצה, מוודא שמספר הגולות מתאים לפי החוקים בעזרת isNotSizeValid

3. מוודא שהגולות שנבחרו שייכות לשחקן הנוכחי בעזרת marblesBelongToPlayer

4. מוודא שהגולות ממוקמות בשורה אחת וצמודות זו לזו בעזרת marblesInLineAndAjacent

5. קובע את סוג המהלך בעזרת determinMoveType

6. מוודא שהמסלול אותו רוצים לעשות תקין בעזרת isPathValid

**determineMoveType:**

1. מוצאים את הכיוון ליעד

2. אם יש גולה אחת וניתן להזיז אותה ישירות ליעד נחזיר MoveType.SINGLE

3. אם אפשר להזיז את כל הגולות צעד לכיוון היעד, מהלך קו אז נחזיר MoveType.INLINE

4. אם אפשר להזיז את הגולות הצידה בכיוון המהלך נחזיר מהלך צד MoveType.SIDESTEP

5. אחרת נחזיר null

**executeMove:**

1. ננקה את הרשימה של הגולות ששימשו את המהלך

2. לפי סוג המהלך נקרא לפונקציה המתאימה,

2.1. מהלך בודד, פשוט נשנה את ערכי שני התאים שהם חלק מהמערך

2.2. מהלך קו, executeInLineOrSingleMove

2.3. מהלך צד, executeSideStepMove

3. נדפיס במידה והייתה בעיה בביצוע המהלך

**undoMove:**

1. נעבור על כל גולה ששימשה את המהלך

1.1. נחזיר את המצב למצב המקורי שלו

## **שם מחלקה: Computer**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **פונקציה** | **פרמטרים** | **מחזירה** | **יעילות** | **תיאור** |
| cellsToMoveTo | ללא פרמטרים | ללא החזרה | O(n) | מסנן את התאים אליהם גולות השחקן יכולים לזוז, בעצם יעדים פוטנציאלים |
| isNeighborOfOpponentCell | Cell | Boolean | O(1) | לכל שכן של תא מסוים בודק אם השכנים שלו ריקים או תפוסים על ידי גולת יריב |
| isNeighborOfPlayerCell | Cell | Boolean | O(1) | לכל שכן של תא מסוים בודק אם השכנים שלו ריקים או תפוסים על ידי גולת שלו |
| updateMarblesList | ללא פרמטרים | ללא החזרה | O(n) | מעדכנת את רשימות הגולות של השחקן והיריב |
| getAllPotentialMovesForBoth | ללא פרמטרים | ללא החזרה | O(n^3) | מעדכנת את רשימת המהלכים האפשריים עבור שני השחקנים על ידי שימוש בפונקציה שמחשבת עבור שחקן ספציפי |
| getAllPotentialMovesForPlayer | שחקן, גולות שחקן, גולות יעדים, רשימת מהלכים | ללא החזרה | O(n^3) | מחשבת את כל המהלכים האפשריים עבור שחקן מסויים |
| computerTurn | ללא פרמטרים | Move | O(n^3) | מחזירה את המהלך הטוב ביותר עבור המחשב, הפונקציה בודקת את כל המהלכים האפשריים, מחשבת ציון עבור כל מהלך, ובוחרת את המהלך עם הציון הגבוה ביותר |
| evaluatesBoardState | Move | double | O(n^2) | מעריכה את מצב המשחק לאחר ביצוע מהלך מסויים, ההערכה מתבססת על מספר גורמים |

**פסיאודו קוד -**

## **שם מחלקה: MenuController**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **פונקציה** | **פרמטרים** | **מחזירה** | **יעילות** | **תיאור** |
| Cell | Int x, int y, int state | ללא החזרה | O(1) | בנאי שמאתחל תא חדש |
| getNeighborsMap | ללא פרמטרים | ללא החזרה | O(1) | מחזירה מפה של תאי השכן של התא עם הכיוונים אליהם |
| getNeighborInDirection | Direction | Cell | O(n) | מחזירה את תא השכן בכיוון נתון |
| isBorder | ללא פרמטרים | Boolean | O(1) | קובעת האם התא הוא תא שפה |

***התוכנית הראשית שמנהלת את המשחק -***

***מדריך למשתמש -***

***רפלקציה -***

***ביבליוגרפיה -***

# A graph paper with drawings Description automatically generatedA graph paper with writing on it Description automatically generated***נספחים -***