## עבודת גמר

## לקבלת תואר טכנאי

# הנדסת תוכנה

### מכללה: אורט חולון

### שם הפרויקט Goumko :

### שם הסטודנט: לירוי נבר

### ת"ז: 328272539

### שם המנחה: הילה ויינשטיין

### 

## תשפ"ד מאי 2024

תוכן עיניים

[1. תקציר: 3](#_Toc165824505)

[2. רקע היסטורי: 4](#_Toc165824506)

[3. תיאור כללי של המשחק 5](#_Toc165824507)

[4. תיאור הבעיה האלגוריתמית 6](#_Toc165824508)

[5. סקירת האלגוריתמים בתחום הבעיה 7](#_Toc165824509)

[6. מושגים 8](#_Toc165824510)

[7. אסטרטגיה 9](#_Toc165824511)

[8. מבנה נתונים 10](#_Toc165824512)

[בגרפיקה: 10](#_Toc165824513)

[בלוגיקה: 10](#_Toc165824514)

[9. UML 11](#_Toc165824515)

[10. ארכיטקטורת Top-Down 12](#_Toc165824516)

[11. תיאור סביבות העבודה ושפת התכנות 15](#_Toc165824517)

[12. תיאור הממשקים 16](#_Toc165824518)

[מסך פתיחה: 16](#_Toc165824519)

[מסך הגדרות: 17](#_Toc165824520)

[מסך חוקים: 18](#_Toc165824521)

[מסך המשחק 19](#_Toc165824522)

[מסך הניצחון 19](#_Toc165824523)

[13. אלגוריתם ראשי 20](#_Toc165824524)

[14. תיאור המחלקות הראשיות 21](#_Toc165824525)

[BoardPanel 21](#_Toc165824526)

[OuterFrame 22](#_Toc165824527)

[Square 26](#_Toc165824528)

[GameManager 29](#_Toc165824529)

[15. מחלקת ראשית 31](#_Toc165824530)

[16. מדריך למשתמש 32](#_Toc165824531)

[17. קוד + תיעוד 33](#_Toc165824532)

[gameLogic 33](#_Toc165824533)

[gameUI 41](#_Toc165824534)

[18. רפלקציה 83](#_Toc165824535)

[19. בביבליוגרפיה 84](#_Toc165824536)

# תקציר:

הפרויקט אשר פיתחתי הוא גרסת מחשב של המשחק הסיני העתיק ג'ומוקו הקלאסי או בשמו ההמוני חמש בשורה. ספר הפרויקט הנ"ל סוקר את מלוא פרויקט הגמר שלי, ואלו הן חלק מן הנקודות המרכזיות שניתן למצוא בספר זה:

* תיאור המשחק ורקע תיאורטי
* ארכיטקטורת המערכת
* אלגוריתם ודרך עבודה של הבינה המלאכותית
* סביבת עבודה, שפה וכלים אשר בשימוש על מנת לפתח את הפרויקט
* הסבר על הפונקציות והאלגוריתמים הראשיים
* הקוד במלואו ובעל תיעוד
* מדריך למשתמש
* רפלקציה אישית על תהליך העבודה
* ביבליוגרפיה ונספחים

# רקע היסטורי:

ג'ומוקו, הידוע גם בשם חמישה בשורה, הוא משחק לוח קלאסי ששוחק מאות שנים. מוצאו מגיע מסין העתיקה, שם נקרא Wu Zi Qi (五子棋) או " חמישה אבנים ". פשטותו ועומק האסטרטגיה של המשחק הפכו אותו לפופולרי עד לימינו ועל גבי מדינות שונות.

היסטורית, ג'ומוקו משוחק על לוח של משחק הגו, שבו מדובר ברשת של קווים חותכים. השחקנים מניחים בתורם את האבנים שלהם על נקודות החיתוך של הקווים, עם המטרה ליצור שורה ברצף של חמישה אבנים אופקית, אנכית או באלכסון. המשחק מחייב תכנון וחשיבה מראש, שכן השחקנים חייבים לצפות את המהלך של היריב וגם לעבוד כדי להשיג ניצחון.

במשך השנים, ג'ומוקו התפתח והשתנה בהתאם לתרבויות וסביבות שונות. הוא משוחק כיום בכמה גרסאות כללים וגדלי לוחות שונים, מלוחות עץ קטנים עד לרשתות גדולות יותר שמצוירות על החול. בשנים האחרונות, ג'ומוקו גם נכנס לעולם הדיגיטלי, עם פלטפורמות מקוונות ואפליקציות שמאפשרות לשחקנים להתמודד נגד יריבים מרחבי העולם.

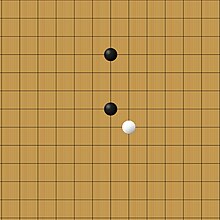
למרות המוצא העתיק שלו, ג'ומוקו נשאר משחק אהוב לאנשים מכל הגילאים, שמרה על השחקנים בזכות השילוב של אסטרטגיה, מיומנות וחשיבה טקטית.





# תיאור כללי של המשחק

לוח המשחק הינו לוח 15X15 משבצות כאשר מניחים אבנים שחורות או לבנות על הצטלבויות המשבצות.



חוקי המשחק:

1. **המטרה**: המטרה בגומוקו היא להיות הראשון שמצליח ליצור רצף של חמישה אבנים ברצף אופקי, אנכי או באלכסון על לוח המשחק.
2. **התור**: המשחק מתחיל עם לוח ריק ושני שחקנים. הם משחקים בתורים תוך הנחת אבן בכל תור.
3. **המהלך**: בתור שלו, כל שחקן מציין את המהלך שלו על לוח המשחק על ידי הנחת אבן על נקודת החיתוך של שני קווים. אחרי שהאבן מונחת, התור מעבר לשחקן השני.
4. **ניצחון**: השחקן שמצליח ליצור רצף של חמישה אבנים ברצף על הלוח, באופן אופקי, אנכי או באלכסון, מנצח במשחק.
5. **תיקו**: אם כל הלוח ממולא באבנים ואין ניצחון ברור, המשחק נגמר בתיקו.

# תיאור הבעיה האלגוריתמית

1. **תיאור הבעיה האלגוריתמית:**

הבעיה האלגוריתמית שלי היא ליצור משחק ממוחשב שמטרתו לנצח במשחק על פי חוקיו.

הבעיה המרכזית טמונה בכתיבת אלגוריתם למתן הצעות מהלכים לשחקנים, הצעת אסטרטגיות אפשריות, וכמובן, קביעת המהלך הטוב ביותר בהתחשב במצב הנוכחי של לוח המשחק.

בנוסף, יש צורך ביישום אלגוריתמי תזמון כדי להבטיח זמני תגובה רציפים ומהירים למשתמשים. כאשר מתמודדים עם בעיה כזו, צריך להגדיר בבירור את כללי המשחק, כולל תנאים לסיום המשחק והמהלכים התקינים ששחקנים יכולים לעשות. יישום אלגוריתמים לזיהוי התקפות והגנות אפשריות יכול גם לשפר את ביצועי הבוט.

# סקירת האלגוריתמים בתחום הבעיה

האלגוריתמים הקיימים לפתירת בעיה זאת הם:

* + - 1. אלגוריתם MINIMAX – האלגוריתם מחשב את המהלך הטוב ביותר לשחקן כדי לנצח במשחק
      2. אלגוריתם Monte Carlo Simulation - יישום הדמיית מונטה קרלו פשוטה לבחירת מהלכים. הבוט מדמה משחקים אקראיים מהמצב הנוכחי ובוחר את המהלך שמתפקד טוב בממוצע.
      3. אלגוריתם Temporal Difference Learning - יישום למידה של הבדלים זמניים (TD) עבור הבוט המשחק. השתמש בטכניקות כגון Q-learning או SARSA כדי לעדכן את הידע של הבוט ולשפר את קבלת ההחלטות שלו לאורך זמן בהתבסס על תגמולים ועונשים.

אלה הם 3 אלגוריתמים שמצאתי לפתירת בעיה אך לא אשתמש בהם אלא אצור אחד בעצמי.

# מושגים

* אוטומט מצבים (State Machine): מודל חישובי שבו התנהגות המערכת או התוכנה מוגדרת על ידי רצפי מצבים נגישים. בכל מצב, המערכת נמצאת במצב מסוים וניתן לבצע פעולות מסוימות אשר עוברות את המערכת למצבים חדשים.
* מערך דו ממדי (Two-dimensional Array): מבנה נתונים בתכנות המכיל ערכים מסודרים בשני מימדים, כלומר רשימה של רשימות. ניתן לגשת לאיבר במערך דו ממדי על ידי ציון של שני אינדקסים, אחד לשורה והשני לעמודה.
* Java Swing: מרכיבי גרפיים של פלטפורמת Java שמשמשים לפיתוח יישומות גרפיות. הם מספקים כלים ליצירת ממשקי משתמש מתקדמים, כולל חלונות, כפתורים, תיבות טקסט, ועוד.
* Java Clip: מחלקה בשפת תכנות Java שמשמשת לניהול קבצי קול. היא מאפשרת להעתיק, לחתוך, ולהדביק נתונים של קול, כמו גם לנגן קבצי קול במחשב.
* Enum (Enumeration): סוג משתנה בשפת תכנות המאפשר להגדיר קבוצה סגורה של ערכים אפשריים למשתנה. ב-Java, כאשר משתמשים ב-Enum, ניתן להגדיר סוג משתנה עם קבוצת ערכים אפשריים קבועה מראש.
* Custom Exception: זוהי חריגה (exception) שמוגדרת על ידי מפתח התוכנה, ואינה נגררת באופן אוטומטי על ידי השפה או הסביבה הרצה. מטרתה לאפשר למפתחי התוכנה לטפל בתקלות מסוימות בצורה מותאמת אישית.
* Events: אירועים בתכנות מציינים פעולות או מצבים שיכולים לגרום לרץ של קוד. כאשר אירוע מתרחש, מופעל קוד מסוים שקשור לאירוע זה. ב-Java, אירועים נמצאים בקשר רב עם Java Swing, ומשמשים לטיפול בפעולות של משתמשים, כגון לחיצות על כפתורים או מעבר בין תאים בטבלאות.

# אסטרטגיה

האסטרטגיה של המשחק של בוט Gomoku

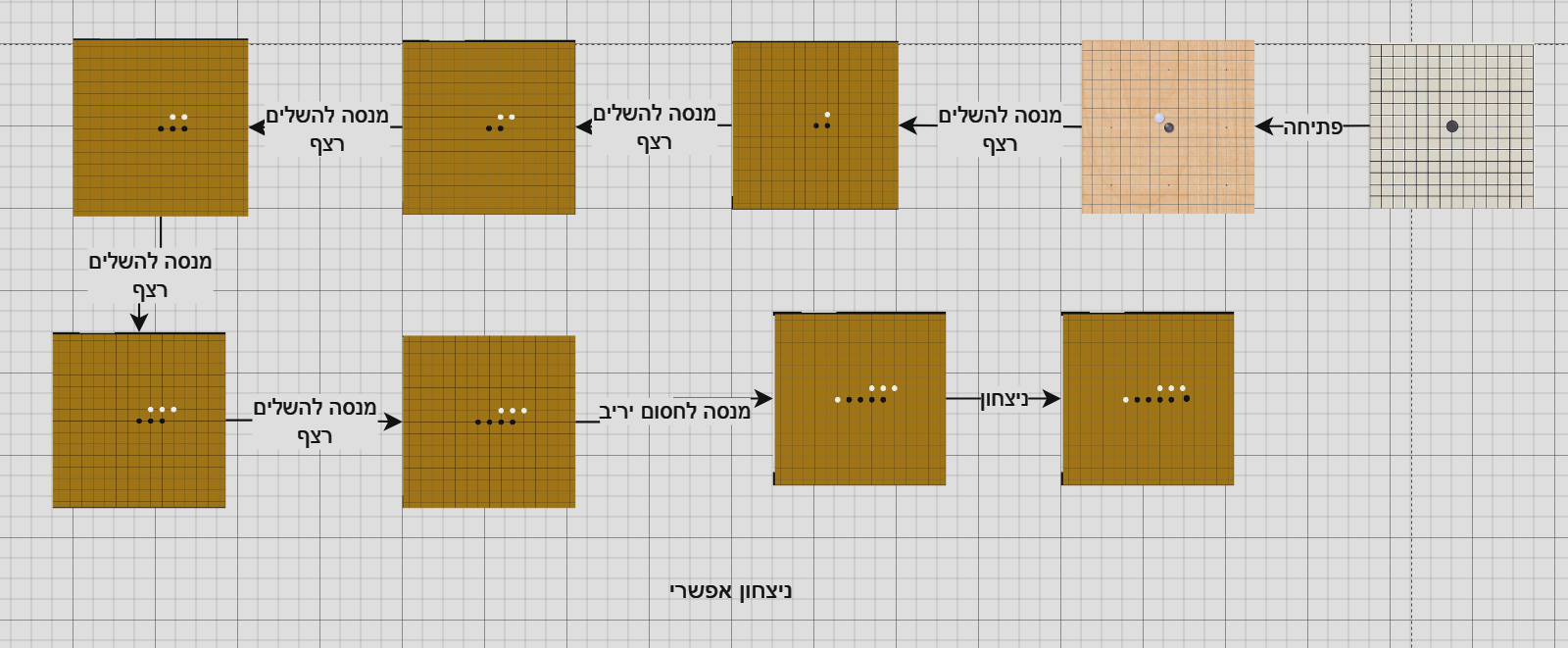
המשחק של הבוט בגומוקו מבוסס על שילוב של חיזיון טקטי ותכנון אסטרטגי.

תחילה, הבוט מעדיף לשלוט במרכז הלוח בשלב הראשון של המשחק כדי להציב יתרון ולהגביל את האפשרויות של היריב. הוא מציב אבנים באופן אסטרטגי כדי ליצור כמה שיותר קווי ניצחון אפשריים ובו זמנית לחסום את ההתקדמות של היריב.

במהלך המשחק, הבוט סורק באופן רציף את הלוח לאיומים מיידיים הנוצרים על ידי היריב. כאשר הוא מזהה רצף של 3 אצל יריב יעבור למצב הגנתי ויחל לחסום משני הכיוונים את היריב ובו זמנית מחפש הזדמנויות ליצור חסימות ואיומים משלו.

וכאסטרטגיה ארוכת טווח, הבוט מעריך מצבי עתיד פוטנציאליים (רצפים של 3/4 הם פוטנציאלי לניצחון) ומעריך את סיכויי ההצלחה בהתבסס על המשך המשחק האפשריים. הוא שומר על איזון בין התקפה להגנה, מתאים את האסטרטגיה שלו בהתאם לדינמיקה המתרחשת במשחק.

בסופו של דבר, המשחק של הבוט בגומוקו מאופיין ביכולתו להעריך ולהגיב לאיומים טקטיים מיידיים ולשיקולים אסטרטגיים ארוכי הטווח, ובכך למקסם את סיכויי ההצלחה שלו נגד מגוון רחב של יריבים.



# מבנה נתונים

## בגרפיקה:

* במחלקת BoardPanel:

קיימת מטריצה בגודל 15X15 המייצגת את הלוח הגרפי של המשחק. המטריצה היא מסוג

המחלקה Square והיא מאותחלת בפעולה הבונה של מחלקת BoardPanel.

* במחלקת Square:

מחלקה המייצגת משבצת על הלוח. למשבצת מספר תכונות שונות המשמשות כדי להבדיל אותה ממשבות אחרות: האם ריקה, השחקן שעל המשבצת, המשבצת עצמה ומיקום המשבצת על הלוח.

## בלוגיקה:

* מחלקת Square:

כמו בגרפיקה, מחלקה זו משמשת כדי לתאר משבצת על הלוח. בשונה מן המחלקה המקבילה

בגרפיקה, אנחנו לא שומרים את מיקום המשבצת מכיוון שאין צורך בכך. המחלקה שומרת רק

את: השחקן שעל המשבצת במידה ויש כזו.

* מחלקת Board:

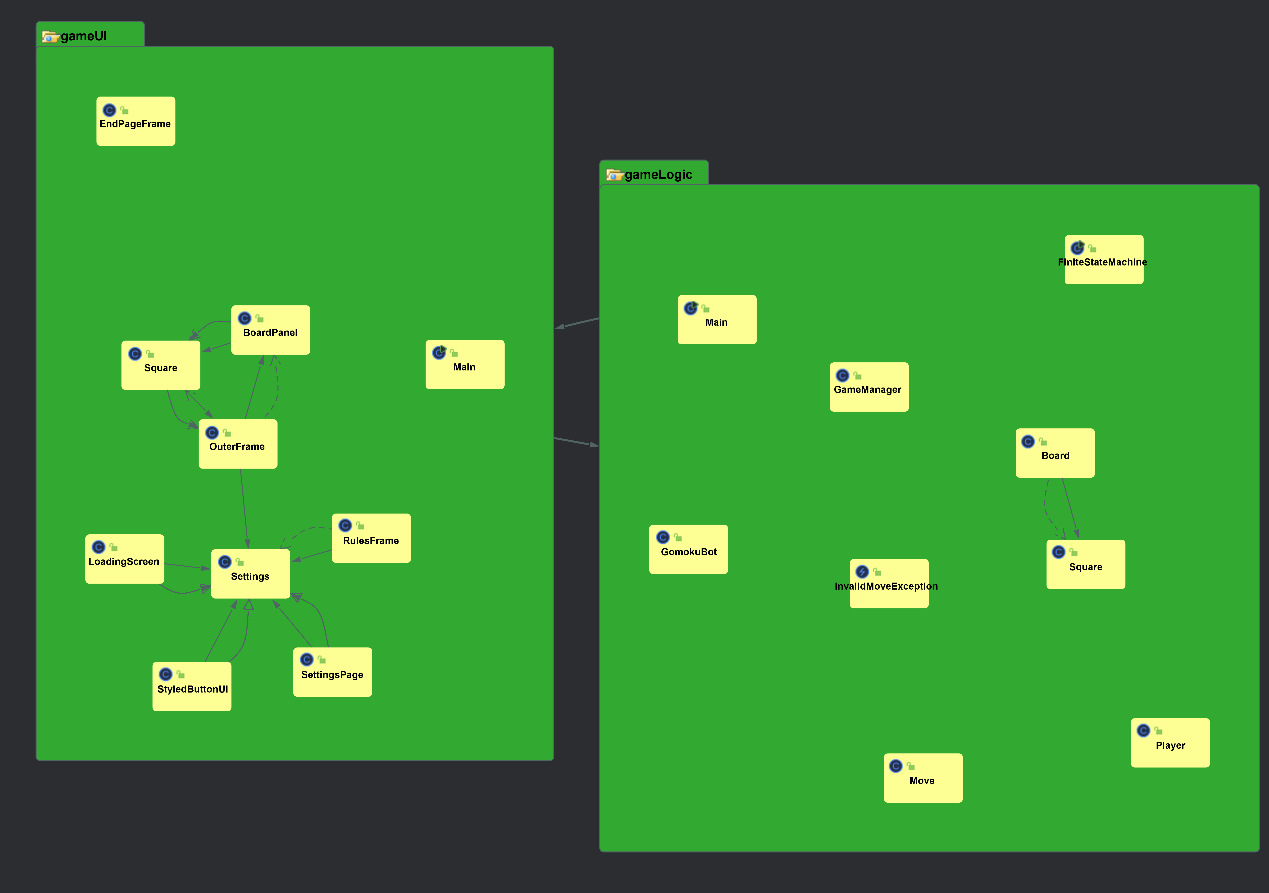
מחלקה אשר מהווה כלוח הלוגי של המשחק. הלוח הלוגי מדמה את הלוח הגרפי והוא בעצם

מייצג מטריצה של 15X15 אשר כל משבצת בה היא מסוג Square.

* מחלקת GomokuBot:

מחלקה המייצגת את האוטומט המהלכים של הבינה המלאכותית.

# UML



# ארכיטקטורת Top-Down

מארזים בפרויקט:

* gameLogic
* gameUI

מחלקות ב:gameLogic

* Board.java
  + Board()
  + clearBoard()
  + get\_board()
  + getSquare(int, int)
  + printBoard()
  + updateBoard(Soldier, int, int)
* GameManager.java
  + GameManager(BoardPanel)
  + isValidIndex(int)
  + isValidMove(int, int)
  + startGame(int, int)
  + winnerCulc()
  + winnerCulcHelper(char)
  + makeMove()
* GameStatus.java > ENUM
  + GameStatus(String)
  + getSymbol()
* InvalidMoveException.java
* Main.java
  + main(String[])
* Player.java
  + Player(Soldier)
  + equals(Object)
  + get\_gameStatus()
  + get\_player()
  + set\_gameStatus(GameStatus)
  + set\_player(Soldier)
* Soldier.java > ENUM
  + Soldier(String)
  + getSymbol()
* Square.java
  + Square()
  + Square(Soldier)
  + clearSquare()
  + get\_Soldier()
  + is\_hasPlayer()
  + toString()
  + updateSquare(Soldier)
* GomokuBot.java
  + GomokuBot(BoardPanel)
  + findAttackMove()
  + findDefenseMove()

מחלקות בgameUI:

* BoardPanel.java
  + BoardPanel(OuterFrame)
  + clearBoard()
  + getPanels()
  + getSquare(int, int)
  + undoLastMove()
* EndPageFrame.java
  + EndPageFrame()
  + setTitleLabelText()
* GameStatus.java > ENUM
* LoadingScreen.java
  + LoadingScreen()
  + startMainFunctionality()
* Main.java
  + main(String[])
  + togglePlayer()
* OuterFrame.java
  + endGame(JFrame)
  + endGameWrapper()
  + turnTimerRestart()
  + updateButton()
  + OuterFrame()
  + actionPerformed(ActionEvent)
  + get\_currSquare()
  + get\_log()
  + get\_Manager()
  + getBoardPanel()
  + loadGame(String)
  + paint(Graphics)
  + removeLastLogLine()
  + restartGame()
  + saveGame()
  + set\_currSquare(Square)
  + set\_log(JTextArea)
  + set\_Manager(GameManager)
  + setBoardPanel(BoardPanel)
* Player.java > ENUM
  + Player(String)
  + getSymbol()
* RulesFrame.java
  + RulesFrame()
* Square.java
  + Square(int, int)
  + clearSquare()
  + get\_i()
  + get\_j()
  + getoFrame()
  + getPlayer()
  + isHasSoldier()
  + paintComponent(Graphics)
  + set\_i(int)
  + set\_j(int)
  + setHasSoldier(boolean)
  + setoFrame(OuterFrame)
  + setPlayer(Player)
* StartPageFrame.java
  + StartPageFrame()
  + loadGame()
  + styleButton(JButton)

# תיאור סביבות העבודה ושפת התכנות

תמונה שמכילה צילום מסך, גרפיקה, טקסט, עיצוב גרפי

התיאור נוצר באופן אוטומטי

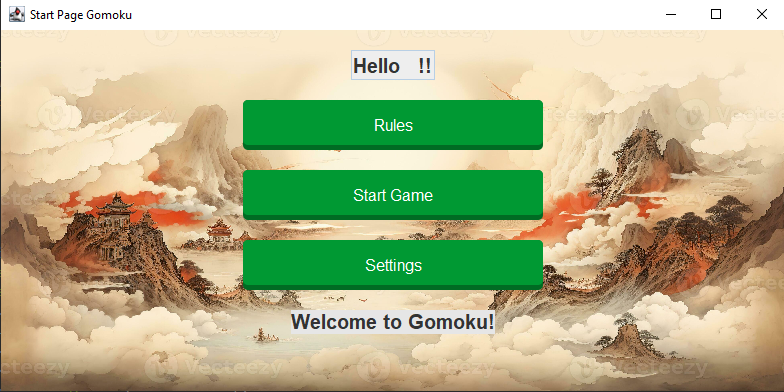
מפרט טכני של המחשב עליו נכתב הפרויקט:

* Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz 2.90 GHz
* 16.0 GB RAM
* סביבת עבודה: Windows 10/11
* שפה: ג'אווה Java
* IDE: Eclipse for Committers 2023

# תיאור הממשקים

## מסך פתיחה:

מסך הפתיחה נפתח במרכז מסך המחשב בגודל 800X400.



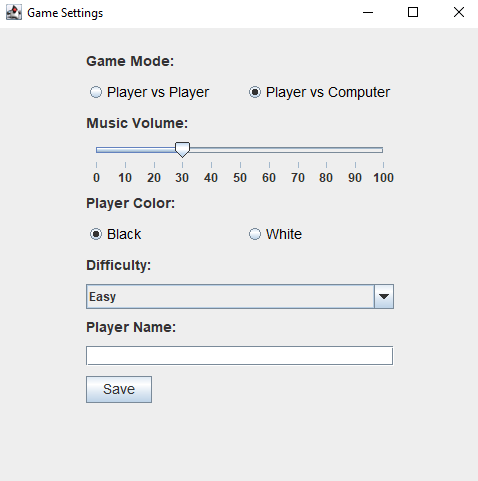
* Rules - מעבר למסך חוקים
* Start Game

לחיצה על כפתור זה תתחיל את המשחק לפני ההגדרות שהמשתמש בחר במסך ההגדרות.

במידה ולא בחר הגדרות שונות יוחלו הגדרות ברירת המחדל שהן:

* + - Mode: PVC
    - Volume: 30
    - Player color: Black
    - Difficulty: Easy
    - Name:
* Settings – מעבר למסך הגדרות

## מסך הגדרות:

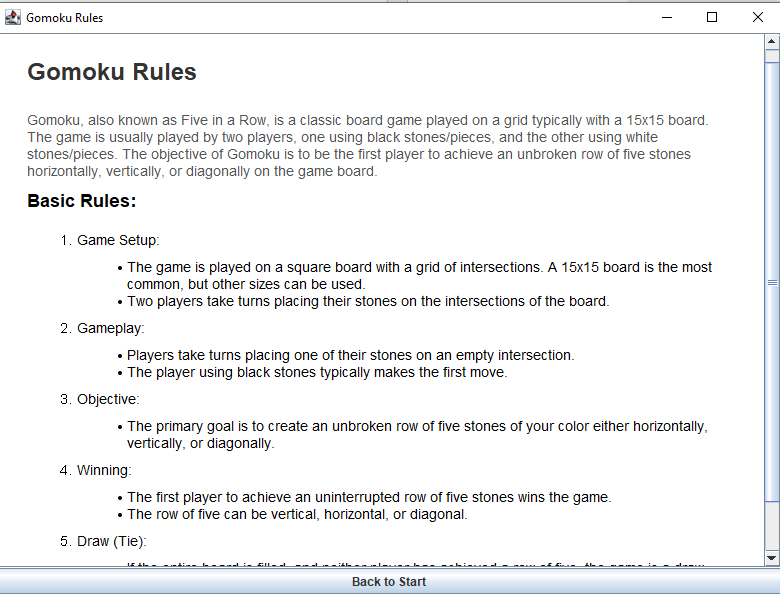


ניתן לבחור ולשנות את ההגדרות לפי בקשת המשתמש

* משחק מול מחשב או מול בן אדם
* ווליום המוזיקה
* צבע השחקן
* רמת קושי
* להכניס את שם המשתמש

בלחיצה על כפתור Save ההגדרות נשמרות והמשתמש מועבר למסך הפתיחה.

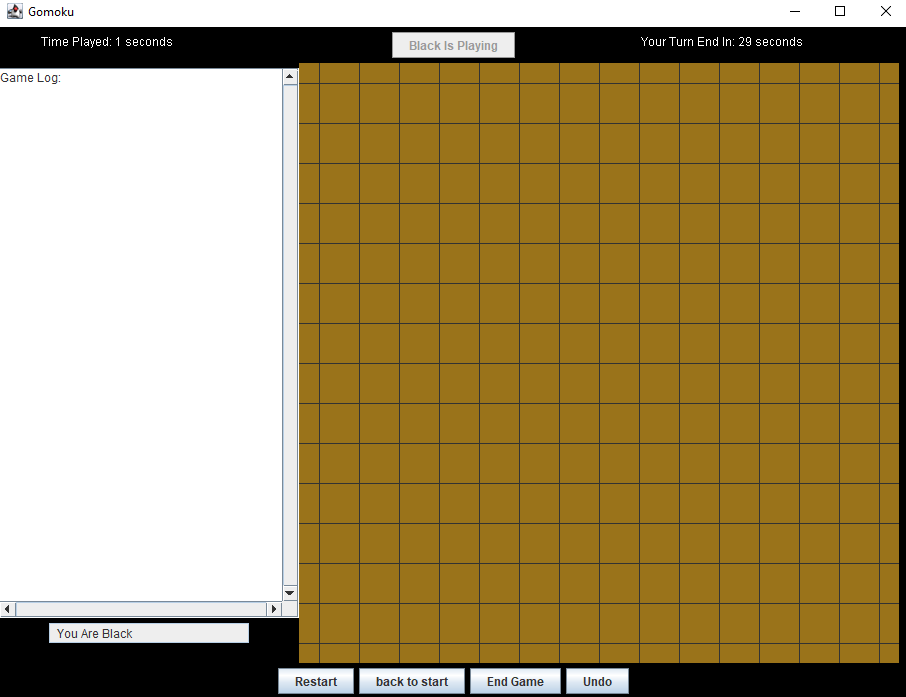
## מסך חוקים:



המסך מציג את חוקי המשחק למשתמש.

בלחיצה על כפתור back to start המשתמש מועבר חזרה למסך הפתיחה.

## מסך המשחק



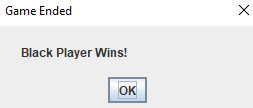
במסך זה קיימים בנוסף למשחק אפשרויות של:

* Restart – התחלה מחדש
* Back to start – חוזר למסך הפתיחה
* End game – סיים משחק
* Undo – ביטול מהלך אחרון

בנוסף קיימים מספר פיצ'רים נוספים והן:

* בצד השמאלי של המסך game log המראה את המהלכים לאורך המשחק
* למעלה כתוב את השחקן שתורו כרגע
* בצד השמאלי העליון קיים טיימר למשחק
* ובצד הימני העליון קיים טיימר לסוף התור (30 שניות) שבסופו יקפוץ חלון המודיע על גמר התור

## מסך הניצחון

חלון קופץ המשתנה בהתאם למנצח/ תיקו.

# אלגוריתם ראשי

# תיאור המחלקות הראשיות

## BoardPanel

**תכונות:**

* **panels:**
  + סוג: מערך של Square.
  + מה התכונה מבצעת: מאפשרת גישה ושינוי לכל ריבוע בלוח המשחק.
* **NUMBER\_OF\_ROWS:**
  + סוג: קבוע מספרי שלמים.
  + מה התכונה מבצעת: מציינת את מספר השורות והעמודות בלוח המשחק.
* **\_oFrame:**
  + סוג: OuterFrame.
  + מה התכונה מבצעת: מכילה קישור לאובייקט מסוג OuterFrame.

**פונקציות:**

1. **BoardPanel(OuterFrame oFrame):**
   * **כניסה:** מקבלת את מקרא ה-OuterFrame כארגומנט.
   * **מה הפונקציה מבצעת:** בונה את לוח המשחק, מאתחלת את המשתנה panels כמערך של ריבועים, יוצרת מידע לכל ריבוע בתוך לוח המשחק ומוסיפה אותם ללוח. גם מגדירה את הפריסה של לוח המשחק וצבע הרקע.
   * **טענת יציאה:** אין.
2. **clearBoard():**
   * **כניסה:** אין.
   * **מה הפונקציה מבצעת:** מנקה את כל ריבועי לוח המשחק.
   * **טענת יציאה:** אין.
3. **getPanels():**
   * **כניסה:** אין.
   * **מה הפונקציה מבצעת:** מחזירה את מערך הריבועים בלוח.
   * **טענת יציאה:** מחזירה את מערך הריבועים בלוח.
4. **getSquare(int i, int j):**
   * **כניסה:** מקבלת שני מספרים שמייצגים את השורה והעמודה בלוח.
   * **מה הפונקציה מבצעת:** מחזירה את הריבוע במיקום הספציפי בלוח.
   * **טענת יציאה:** מחזירה את הריבוע במיקום הספציפי בלוח.
5. **undoLastMove():**
   * **כניסה:** אין.
   * **מה הפונקציה מבצעת:** בודקת אם קיימת מקום אחרון שבו הונח לוחם ואם כן, מוחקת את הלוחם מהריבוע המתאים בלוח המשחק.
   * **טענת יציאה:** מחזירה true אם בוצעה בהצלחה השחזור, וfalse אם לא נמצאה כלי שניתן לשחזר במהלך האחרון.

## OuterFrame

**תכונות:**

* **playerButton:**
  + סוג: JButton
  + תפקיד: מציג את שם השחקן התורן.
* **restart:**
  + סוג: JButton
  + תפקיד: מפעיל את אתחול המשחק מחדש.
* **back:**
  + סוג: JButton
  + תפקיד: מאפשר חזרה לדף הראשי של המשחק.
* **saveButton:**
  + סוג: JButton
  + תפקיד: מאפשר שמירת המשחק הנוכחי.
* **boardPanel:**
  + סוג: BoardPanel
  + תפקיד: מציג את לוח המשחק ואת הקלטות המשתמש.
* **timer:**
  + סוג: JLabel
  + תפקיד: מציג את זמן המשחק הכולל.
* **turnTimer:**
  + סוג: JLabel
  + תפקיד: מציג את זמן תור השחקן הנוכחי.
* **\_Manager:**
  + סוג: GameManager
  + תפקיד: מנהל את המשחק, כולל לוח המשחק והלוגיקה.
* **\_currSquare:**
  + סוג: Square
  + תפקיד: מתייחס לתא הנוכחי שנבחר על ידי השחקן.
* **\_frame:**
  + סוג: JFrame
  + תפקיד: מסגרת החלון הראשית של המשחק.
* **\_log:**
  + סוג: JTextArea
  + תפקיד: מציג את היסטוריית המשחק (לוג).
* **clip:**
  + סוג: AudioClip
  + תפקיד: משמיע צלילי המשחק.

**פונקציות:**

1. OuterFrame():
   * **טענת כניסה:** אין.
   * **מה הפונקציה מבצעת:** בונה את ממשק המשתמש של המשחק, עם כל הכפתורים והלוח המתאים.
   * **טענת יציאה:** אין.
   * **יעילות:** O(1).
2. updateButton():

* **טענת כניסה:** אין.
* **מה הפונקציה מבצעת:** מעדכנת את הטקסט על כפתור השחקן הנוכחי.
* **טענת יציאה:** אין.
* **יעילות:** O(1).

1. restartGame():

* **טענת כניסה:** אין.
* **מה הפונקציה מבצעת:** מאפסת את הלוח ומתחילה את המשחק מחדש.
* **טענת יציאה:** אין.
* **יעילות:** O(1).

1. paint(Graphics g):

* **טענת כניסה:** אין.
* **מה הפונקציה מבצעת:** מציירת על גבי המסגרת טקסט עם הזמן שעבר וזמן תור השחקן הנוכחי.
* **טענת יציאה:** אין.
* **יעילות:** O(1).

1. actionPerformed(ActionEvent e):

* **טענת כניסה:** אין.
* **מה הפונקציה מבצעת:** טיפול בארוע של טיימר - מעדכנת את הזמן הנוכחי ומציירת מחדש את המסגרת.
* **טענת יציאה:** אין.
* **יעילות:** O(1).

1. turnTimerRestart():

* **טענת כניסה:** אין.
* **מה הפונקציה מבצעת:** מאפסת את הזמן לתור הנוכחי.
* **טענת יציאה:** אין.
* **יעילות:** O(1).

1. endGame(JFrame frame):

* **טענת כניסה:** מקבלת את המסגרת הנוכחית של המשחק.
* **מה הפונקציה מבצעת:** מסיימת את המשחק ומציגה הודעה על תוצאת המשחק.
* **טענת יציאה:** אין.
* **יעילות:** O(1).

1. endGameWrapper():

* **טענת כניסה:** אין.
* **מה הפונקציה מבצעת:** קוראת לפונקציה endGame() עם המסגרת הנוכחית.
* **טענת יציאה:** אין.
* **יעילות:** O(1).

1. get\_currSquare():

* **טענת כניסה:** אין.
* **מה הפונקציה מבצעת:** מחזירה את התא הנוכחי שנבחר על ידי השחקן.
* **טענת יציאה:** התא הנוכחי שנבחר על ידי השחקן.
* **יעילות:** O(1).

1. set\_currSquare(Square \_currSquare):

* **טענת כניסה:** מקבלת אובייקט Square כארגומנט.
* **מה הפונקציה מבצעת:** מגדירה את התא הנוכחי שנבחר על ידי השחקן.
* **טענת יציאה:** אין.
* **יעילות:** O(1).

1. get\_Manager():

* **טענת כניסה:** אין.
* **מה הפונקציה מבצעת:** מחזירה את מנהל המשחק.
* **טענת יציאה:** מנהל המשחק.
* **יעילות:** O(1).

1. set\_Manager(GameManager \_Manager):

* **טענת כניסה:** מקבלת אובייקט GameManager כארגומנט.
* **מה הפונקציה מבצעת:** מגדירה את מנהל המשחק.
* **טענת יציאה:** אין.
* **יעילות:** O(1).

1. get\_log():

* **טענת כניסה:** אין.
* **מה הפונקציה מבצעת:** מחזירה את תיבת הטקסט של לוג המשחק.
* **טענת יציאה:** תיבת הטקסט של לוג המשחק.
* **יעילות:** O(1).

1. set\_log(JTextArea \_log):
   * **טענת כניסה:** מקבלת אובייקט JTextArea כארגומנט.
   * **מה הפונקציה מבצעת:** מגדירה את תיבת הטקסט של לוג המשחק.
   * **טענת יציאה:** אין.
   * **יעילות:** O(1).
2. removeLastLogLine():
   * **טענת כניסה:** אין.
   * **מה הפונקציה מבצעת:** מסירה את השורה האחרונה בלוג המשחק.
   * **טענת יציאה:** אין.
   * **יעילות:** O(1).
3. getBoardPanel():
   * **טענת כניסה:** אין.
   * **מה הפונקציה מבצעת:** מחזירה את לוח המשחק.
   * **טענת יציאה:** לוח המשחק.
   * **יעילות:** O(1).
4. setBoardPanel(BoardPanel boardPanel):
   * **טענת כניסה:** מקבלת אובייקט BoardPanel כארגומנט.
   * **מה הפונקציה מבצעת:** מגדירה את לוח המשחק.
   * **טענת יציאה:** אין.
   * **יעילות:** O(1).

## Square

**תכונות:**

* **player:**
* **סוג:** Player
* **תפקיד:** מציין את השחקן המחזיק בריבוע (שחור, לבן, או ריק(
* **hasSoldier:**
  + **סוג:**  boolean
  + **תפקיד:** מציין האם בריבוע יש חייל.
* **\_i, \_j:**
  + **סוג:**  int
  + **תפקיד:** מיקום הריבוע בלוח (שורה ועמודה).
* **oFrame:**
  + **סוג:**  OuterFrame
  + **תפקיד:** קישור למסגרת החיצונית (OuterFrame) שבה מופיע הריבוע.
* **\_currSquare:**
  + **סוג** Square **:**
  + **תפקיד:** מצביע לעצמו (שימושי בתוך המחלקה).

**פונקציות:**

1. Square (constructor):
   * **טענת כניסה:** לא נמצאה.
   * **מה הפונקציה מבצעת:** פונקציה בונה שמאתחלת את המשתנים המרכזיים של הריבוע.
   * **טענת יציאה:** לא נמצאה.
   * **יעילות:** O(1).
2. paintComponent:
   * **טענת כניסה:** לא נמצאה.
   * **מה הפונקציה מבצעת:** מציירת את הריבוע ואת החייל בתוכו (אם יש), בהתאם למצב הנוכחי של הריבוע.
   * **טענת יציאה:** לא נמצאה.
   * **יעילות:** O(1).
3. clearSquare:
   * **טענת כניסה:** לא נמצאה.
   * **מה הפונקציה מבצעת:** משנה את המצב של הריבוע ללא חייל ומציירת מחדש את הריבוע.
   * **טענת יציאה:** לא נמצאה.
   * **יעילות:** O(1).
4. get\_i:
   * **טענת כניסה:** לא נמצאה.
   * **מה הפונקציה מבצעת:** מחזירה את הערך של ה-i.
   * **טענת יציאה:** הערך של ה-i.
   * **יעילות:** O(1).
5. set\_i:
   * **טענת כניסה:** ערך מספרי i.
   * **מה הפונקציה מבצעת:** מקבלת ערך מספרי i ומעדכנת את הערך של ה-i.
   * **טענת יציאה:** לא נמצאה.
   * **יעילות:** O(1).
6. get\_j:
   * **טענת כניסה:** לא נמצאה.
   * **מה הפונקציה מבצעת:** מחזירה את הערך של ה-j.
   * **טענת יציאה:** הערך של ה-j.
   * **יעילות:** O(1).
7. set\_j:
   * **טענת כניסה:** ערך מספרי j.
   * **מה הפונקציה מבצעת:** מקבלת ערך מספרי j ומעדכנת את הערך של ה-j.
   * **טענת יציאה:** לא נמצאה.
   * **יעילות:** O(1).
8. getoFrame:
   * **טענת כניסה:** לא נמצאה.
   * **מה הפונקציה מבצעת:** מחזירה את הערך של ה-oFrame.
   * **טענת יציאה:** הערך של ה-oFrame.
   * **יעילות:** O(1).
9. setoFrame:
   * **טענת כניסה:** ערך מסוג OuterFrame.
   * **מה הפונקציה מבצעת:** מקבלת ערך מסוג OuterFrame ומעדכנת את ה-oFrame.
   * **טענת יציאה:** לא נמצאה.
   * **יעילות:** O(1).
10. isHasSoldier:
    * **טענת כניסה:** לא נמצאה.
    * **מה הפונקציה מבצעת:** מחזירה את הערך של ה-hasSoldier.
    * **טענת יציאה:** הערך של ה-hasSoldier.
    * **יעילות:** O(1).
11. setHasSoldier:
    * **טענת כניסה:** ערך בוליאני hasSoldier.
    * **מה הפונקציה מבצעת:** מקבלת ערך בוליאני hasSoldier ומעדכנת את ה-hasSoldier.
    * **טענת יציאה:** לא נמצאה.
    * **יעילות:** O(1).
12. getPlayer:
    * **טענת כניסה:** לא נמצאה.
    * **מה הפונקציה מבצעת:** מחזירה את השחקן של הריבוע.
    * **טענת יציאה:** השחקן של הריבוע.
    * **יעילות:** O(1).
13. setPlayer:
    * **טענת כניסה:** ערך מסוג Player.
    * **מה הפונקציה מבצעת:** מקבלת ערך מסוג Player ומעדכנת את השחקן של הריבוע.
    * **טענת יציאה:** לא נמצאה.
    * **יעילות:** O(1).

## GameManager

**תכונות:**

* **\_board:**
  + **סוג:** BoardPanel
  + **תפקיד:** לוח המשחק שבו מתבצעות הפעולות.
* **sequence\_of\_wins:**
  + **סוג:**  int
  + **תפקיד:** מספר החיילים הדרוש ברצף לניצחון (5).

**פונקציות:**

1. GameManager(BoardPanel board):
   * **טענת כניסה:** מקבל BoardPanel ומאתחל את מנהל המשחק עם הלוח הזה.
   * **מה הפונקציה מבצעת:** מאתחלת את המשחק עם לוח המשחק המסופק.
   * **טענת יציאה:** אין.
   * **יעילות:** O(1).
2. boolean isValidMove(int i, int j) throws InvalidMoveException:
   * **טענת כניסה:** מקבלת שני אינדקסים של שורה ועמודה ובודקת אם המהלך חוקי על הלוח המסופק.
   * **מה הפונקציה מבצעת:** בודקת אם המהלך הוא בתחום המותר, אם התא לא כבר כולל חייל, ואם החייל במקום שנבחר הוא של השחקן הנכון.
   * **טענת יציאה:** מחזירה true אם המהלך חוקי, אחרת מזריקה חריגת InvalidMoveException עם הודעת השגיאה המתאימה.
   * **יעילות:** O(1).
3. void winnerCulc():
   * **טענת כניסה:** אין.
   * **מה הפונקציה מבצעת:** בודקת את המצב הנוכחי של המשחק כדי לקבוע אם יש מנצח או תיקו.
   * **טענת יציאה:** אין.
   * **יעילות:** O(n^2).
4. winnerCulcHelper(char checkType):

* **טענת כניסה:** מקבלת תו המציין את סוג הבדיקה (למשל, 'X' או 'O').
* **מה הפונקציה מבצעת:** פונקציה פנימית עוזרת ל-winnerCulc(), בודקת שורות, עמודות ואלכסונים באורך הרצף הדרוש לניצחון (sequence\_of\_wins) כדי למצוא שחקן מנצח.
* **טענת יציאה:** אין.
* **יעילות:** O(n).

1. void placeSoldier(int i, int j):

* **טענת כניסה:** מקבלת שני אינדקסים של שורה ועמודה ומבצעת את המהלך על הלוח.
* **מה הפונקציה מבצעת:** מכניס חייל של השחקן הנוכחי לריבוע שנבחר, מעדכן את לוח המשחק, וקורא לפונקציות winnerCulc ו-updateButton.
* **טענת יציאה:** אין.
* **יעילות:** O(1).

1. void changeTurn():

* **טענת כניסה:** אין.
* **מה הפונקציה מבצעת:** מחליף את השחקן הנוכחי, מעדכן את כפתור השחקן הנוכחי ומאפס את טיימר התור.
* **טענת יציאה:** אין.
* **יעילות:** O(1).

1. String getCurrentPlayer():

* **טענת כניסה:** אין.
* **מה הפונקציה מבצעת:** מחזירה את שם השחקן הנוכחי.
* **טענת יציאה:** שם השחקן הנוכחי.
* **יעילות:** O(1).

1. void restartGame():

* **טענת כניסה:** אין.
* **מה הפונקציה מבצעת:** מאפסת את לוח המשחק, מחליפה את השחקן הנוכחי לחזרה לשחקן הראשון, ומעדכנת את כפתורי השחקן הנוכחי וטיימר המשחק.
* **טענת יציאה:** אין.
* **יעילות:** O(n^2).

# מחלקת ראשית

package gameUI;

public class Main {

public static gameLogic.*GameStatus* *GAME\_STATUS* = gameLogic.*GameStatus*.***in\_progress***;

public static *Player* *currentPlayer* = *Player*.***WHITE***;

public static *Player* togglePlayer() {

*currentPlayer* = (*currentPlayer* == *Player*.***BLACK***) ? *Player*.***WHITE*** : *Player*.***BLACK***;

OuterFrame.*updateButton*();

OuterFrame.*turnTimerRestart*();

return *currentPlayer*;

}

public static void main(String[] args) {

StartPageFrame startPage = new StartPageFrame();

}

}

# מדריך למשתמש

**שלב א': התקנת המשחק**

יש להוריד את תיקיית הפרויקט Project למקום נבחר במחשב.

**שלב ב': הוראות הפעלה**

יש לפתוח את הפרויקט בעזרת eclipse מהמקום שבו התיקייה נשמרה במחשב, להיכנס למחלקה main שבמארז הבסיס ולהריץ את התוכנית.

**שלב ג': הודעות מערכת**

ישנן הודעות קופצות אשר מודיעות לשחקן כאשר נגמר לו התור וכאשר נגמר המשחק.

# קוד + תיעוד

## gameLogic

##### Board

package gameLogic;

public class Board {

private Square[][] \_board;

public Board() {

\_board = new Square[15][15];

int count = 0;

boolean flag = true;

for (int i = 0; i < \_board.length; i++) {

for (int j = 0; j < \_board.length; j++) {

\_board[i][j] = new Square();

}

}

}

/\*

\* Input: N/A Output: N/A Description: Clear The Board From Soldiers

\*/

public void clearBoard() {

for (int i = 0; i < \_board.length; i++) {

for (int j = 0; j < \_board.length; j++) {

\_board[i][j].clearSquare();

}

}

}

/\*

\* Input: N/A Output: N/A Description: Print the current state of the board to

\* the console

\*/

public void printBoard() {

for (int i = 0; i < \_board.length; i++) {

for (int j = 0; j < \_board.length; j++) {

System.***out***.printf("[%s]", \_board[i][j].toString());

}

System.***out***.println("\n");

}

System.***out***.println("---------------------------------------------");

}

/\*

\* Input: N/A Output: The current board Description: Get the current state of

\* the board

\*/

public Square[][] get\_board() {

return \_board;

}

/\*

\* Input: Row index, Column index Output: The square at the specified position

\* Description: Get the square at the specified position on the board

\*/

public Square getSquare(int i, int j) {

return \_board[i][j];

}

/\*

\* Input: Soldier type, Row index, Column index Output: N/A Description: Update

\* the board with the specified soldier type at the specified position

\*/

public void updateBoard(*Soldier* p, int row, int col) {

\_board[row][col].updateSquare(p);

printBoard();

}

}

##### GameStatus

package gameLogic;

public enum *GameStatus* {

***white\_player\_win***("White Win!"),

***black\_player\_win***("Black Win!"),

***draw***("It's A Draw!"),

***in\_progress***(" ");

private final String symbol;

private GameStatus(String symbol) {

this.symbol = symbol;

}

public String getSymbol() {

return symbol;

}

}

##### InvalidMoveException

package gameLogic;

public class InvalidMoveException extends IllegalArgumentException{

public InvalidMoveException() {

super();

}

public InvalidMoveException(String message) {

super(message);

}

}

##### Main

package gameLogic;

import java.util.Scanner;

public class Main {

public boolean currPlayer; // true-> black | false-> white

public static void main(String[] args) {

GameManager GM = new GameManager();

Scanner s = new Scanner(System.***in***);

}

}

##### Move

package gameLogic;

public class Move {

private final byte row;

private final byte col;

public Move(byte row, byte col) {

this.row = row;

this.col = col;

}

public byte getRow() {

return row;

}

public byte getCol() {

return col;

}

}

##### Player

package gameLogic;

public class Player {

private *Soldier* \_player;

private *GameStatus* \_gameStatus;

public Player(*Soldier* p) {

\_player = p;

}

public *Soldier* get\_player() {

return \_player;

}

public void set\_player(*Soldier* \_player) {

this.\_player = \_player;

}

public *GameStatus* get\_gameStatus() {

return \_gameStatus;

}

public void set\_gameStatus(*GameStatus* \_gameStatus) {

this.\_gameStatus = \_gameStatus;

}

*@Override*

public boolean equals(Object obj) {

if (obj == null || !(obj instanceof Player))

return false;

else{

Player s = (Player)obj;

return s.\_player.equals(\_player);

}

}

}

##### Soldier

package gameLogic;

public enum *Soldier* {

***BLACK***("B"),

***WHITE***("W"),

***EMPTY***("E");

private final String symbol;

private Soldier(String symbol) {

this.symbol = symbol;

}

public String getSymbol() {

return symbol;

}

}

##### Square

package gameLogic;

public class Square {

private *Soldier* \_Soldier;

private boolean \_hasPlayer;

public Square() {

\_Soldier = *Soldier*.***EMPTY***;

\_hasPlayer = false;

}

public Square(*Soldier* s) {

\_Soldier = s;

\_hasPlayer = false;

}

/\* Input: N/A

\* Output: The current soldier on the square

\* Description: Get the current soldier on the square

\*/

public *Soldier* get\_Soldier() {

return \_Soldier;

}

/\* Input: N/A

\* Output: True if the square has a player, false otherwise

\* Description: Check if the square has a player

\*/

public boolean is\_hasPlayer() {

return \_hasPlayer;

}

/\* Input: N/A

\* Output: N/A

\* Description: Clear the square from soldiers and mark it as not having a player

\*/

public void clearSquare() {

\_Soldier = *Soldier*.***EMPTY***;

\_hasPlayer = false;

}

/\* Input: Soldier type

\* Output: N/A

\* Description: Update the square with the specified soldier type and mark it as having a player

\*/

public void updateSquare(*Soldier* s) {

\_Soldier = s;

\_hasPlayer = true;

}

*@Override*

public String toString() {

return (\_Soldier == *Soldier*.***BLACK***) ? "B" : (\_Soldier == *Soldier*.***WHITE***) ? "W" : "E";

}

}

##### State

package gameLogic;

public enum *State* {

***NO\_SEQ***(true, "Attacking"), ***HAS\_SEQ***(false, "Defending");

private final boolean explicit;

private final String description;

State(boolean explicit, String description) {

this.explicit = explicit;

this.description = description;

}

}

## gameUI

##### BoardPanel

package gameUI;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Graphics;

import java.awt.GridLayout;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileOutputStream;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import javax.swing.BorderFactory;

import javax.swing.JPanel;

import gameLogic.Soldier;

public class BoardPanel extends JPanel {

private Square[][] panels;

public static final int ***NUMBER\_OF\_ROWS*** =15;

private OuterFrame \_oFrame;

public BoardPanel(OuterFrame oFrame) {

super();

\_oFrame = oFrame;

panels = new Square[***NUMBER\_OF\_ROWS***][***NUMBER\_OF\_ROWS***];

setLayout(new GridLayout(***NUMBER\_OF\_ROWS***, ***NUMBER\_OF\_ROWS***));

setBackground(new Color(161, 121, 51));

for (int i = 0; i < panels.length; i++) {

for (int j = 0; j < panels[i].length; j++) {

panels[i][j] = new Square(i,j);

panels[i][j].setoFrame(oFrame);

add(panels[i][j]);

}

}

setPreferredSize(new Dimension(600, 600));

}

public void clearBoard() {

for (int i = 0; i < panels.length; i++) {

for (int j = 0; j < panels[i].length; j++) {

panels[i][j].clearSquare();

}

}

}

public Square[][] getPanels() {

return panels;

}

public Square getSquare(int i,int j) {

return panels[i][j];

}

public boolean undoLastMove() {

// Find the last move and clear the square if a soldier is present

if (\_oFrame.get\_currSquare() != null) {

int i = \_oFrame.get\_currSquare().get\_i();

int j = \_oFrame.get\_currSquare().get\_j();

if (i >= 0 && i < ***NUMBER\_OF\_ROWS*** && j >= 0 && j < ***NUMBER\_OF\_ROWS***) {

if (panels[i][j].isHasSoldier()) {

panels[i][j].clearSquare();

return true;

}

}

}

return false;

}

public void placeMove(int row, int col, *Soldier* black) {

panels[row][col].botPlacePlayer(row,col);

}

}

##### GameStatus

package gameUI;

public enum *GameStatus* {

***white\_player\_win***,

***black\_player\_win***,

***draw***;

}

##### LoadingScreen

package gameUI;

import javax.swing.\*;

import java.awt.\*;

import java.util.Timer;

import java.util.TimerTask;

public class LoadingScreen extends JFrame {

private JProgressBar progressBar;

private Settings \_settings;

public LoadingScreen(Settings s) {

setTitle("Loading");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(300, 100);

setLocationRelativeTo(null); // Center the frame on the screen

\_settings = s;

progressBar = new JProgressBar();

progressBar.setStringPainted(true);

add(progressBar, BorderLayout.CENTER);

// Simulate loading process using a TimerTask

Timer timer = new Timer();

timer.schedule(new ProgressTask(), 0, 100); // Update progress every 100 milliseconds

setVisible(true);

}

class ProgressTask extends TimerTask {

private int progress = 0;

private int currentMessageIndex = 0;

public void run() {

if (progress < 100) {

// Increment progress

progress += 5; // Increment progress by 5%

progressBar.setValue(progress);

// Change progress message every 25%

if (progress % 25 == 0 && currentMessageIndex < 3) {

currentMessageIndex++;

}

String[] progressMessages = {"Loading " + progress + "%", "Still Loading " + progress + "%", "Almost There " + progress + "%", "Finishing Up " + progress + "%"};

progressBar.setString(progressMessages[currentMessageIndex]);

} else {

// Loading complete, close the loading screen and start the main functionality

dispose(); // Close the loading screen

// Start your main functionality here

startMainFunctionality();

cancel(); // Stop the timer task

}

}

}

// Method to start the main functionality of your application

private void startMainFunctionality() {

OuterFrame oFrame = new OuterFrame(\_settings);

dispose();

}

}

##### OuterFrame

package gameUI;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Graphics;

import java.awt.GridBagLayout;

import java.awt.GridLayout;

import java.awt.Image;

import java.awt.Panel;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.text.MessageFormat;

import javax.sound.sampled.AudioInputStream;

import javax.sound.sampled.AudioSystem;

import javax.sound.sampled.Clip;

import javax.sound.sampled.FloatControl;

import javax.sound.sampled.LineUnavailableException;

import javax.sound.sampled.UnsupportedAudioFileException;

import javax.swing.Icon;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTextArea;

import javax.swing.JTextField;

import javax.swing.JTextPane;

import javax.swing.Timer;

import javax.swing.text.StyledEditorKit.BoldAction;

import gameLogic.GameManager;

import gameLogic.GomokuBot;

public class OuterFrame extends JFrame implements ActionListener {

static JButton playerButton;

private JButton restart;

private JButton back;

private JButton saveButton;

private BoardPanel boardPanel;

private Timer timer;

private static Timer turnTimer;

private static Integer turnTimeCount;

private Integer timeCount;

private transient GameManager \_Manager;

private Square \_currSquare;

private static JFrame \_frame;

private JTextArea \_log;

private static Clip clip;

private Settings \_settings;

private GomokuBot \_bot;

private boolean \_turn; // true - bot | false - player

public OuterFrame(Settings settings) {

super("Gomoku");

\_turn = true;

\_settings = settings;

\_currSquare = new Square(-1, -1);

timer = new Timer(1000, this);

turnTimeCount = 30;

turnTimer = new Timer(1000, new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

if (turnTimeCount == 0) {

Main.togglePlayer();

updateButton();

JOptionPane.showMessageDialog(rootPane, "Your Turn Has Ended");

turnTimeCount = 30;

}

turnTimeCount--;

repaint();

}

});

\_frame = this;

timer.start();

turnTimer.start();

\_log = new JTextArea("Game Log: \n");

JPanel panel1 = new JPanel();

JPanel panel2 = new JPanel();

JScrollPane scroll = new JScrollPane(\_log, JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS,

JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);

scroll.setPreferredSize(new Dimension(300, 550));

panel2.add(scroll);

JPanel panel3 = new JPanel();

JPanel panel4 = new JPanel();

JPanel panel5 = new JPanel(new GridBagLayout());

panel2.setPreferredSize(new Dimension(300, 550));

if (boardPanel == null)

boardPanel = new BoardPanel(this);

\_Manager = new GameManager(boardPanel);

playerButton = new JButton();

playerButton.setText((Main.currentPlayer == Player.BLACK ? "White " : "Black ") + "Is Playing");

playerButton.setEnabled(false);

restart = new JButton("Restart");

restart.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

restartGame();

}

});

back = new JButton("back to start");

back.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

clip.stop();

clip.close();

StartPageFrame st = new StartPageFrame(\_settings);

dispose();

turnTimer.stop();

}

});

JTextField \_name = new JTextField(\_settings.getPlayerName() + " You Are " + \_settings.getPlayerColor());

\_name.setPreferredSize(new Dimension(200, 20));

\_name.setEditable(false);

panel2.add(\_name);

timeCount = 0;

panel1.setBackground(Color.BLACK);

panel2.setBackground(Color.BLACK);

panel3.setBackground(Color.BLACK);

panel4.setBackground(Color.BLACK);

panel1.add(playerButton);

panel5.add(boardPanel);

panel4.add(restart);

panel4.add(back);

JButton end = new JButton("End Game");

end.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

endGame(\_frame);

}

});

panel4.add(end);

JButton undoButton = new JButton("Undo");

undoButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

if (boardPanel.undoLastMove()) {

\_log.append(Main.currentPlayer + " player has remove player from:" + \_currSquare.get\_i() + ","

+ \_currSquare.get\_j() + "\n");

Main.currentPlayer = (Main.currentPlayer == Player.BLACK ? Main.currentPlayer = Player.WHITE

: Player.BLACK);

}

}

});

panel4.add(undoButton);

getContentPane().setLayout(new BorderLayout(0, 0));

getContentPane().add(panel1, BorderLayout.NORTH);

getContentPane().add(panel2, BorderLayout.WEST);

getContentPane().add(panel3, BorderLayout.EAST);

getContentPane().add(panel4, BorderLayout.SOUTH);

getContentPane().add(panel5, BorderLayout.CENTER);

\_Manager.startGame(\_currSquare.get\_i(), \_currSquare.get\_j());

pack();

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

setVisible(true);

System.out.println(this.\_settings.toString());

try {

AudioInputStream audioInputStream = AudioSystem.getAudioInputStream(new File("music.wav"));

clip = AudioSystem.getClip();

clip.open(audioInputStream);

// Get the control for volume

FloatControl volumeControl = (FloatControl) clip.getControl(FloatControl.Type.MASTER\_GAIN);

// Set the desired volume level (between 0.0 and 1.0)

float volume = (float) \_settings.getMusicVolume() / 100;

// Calculate the volume level in decibels

float volumeLevel = volumeControl.getMinimum()

+ (volumeControl.getMaximum() - volumeControl.getMinimum()) \* volume;

// Set the volume level

volumeControl.setValue(volumeLevel);

// Loop the audio continuously

clip.loop(Clip.LOOP\_CONTINUOUSLY);

} catch (UnsupportedAudioFileException | IOException | LineUnavailableException ex) {

ex.printStackTrace();

}

// Play the audio when the frame is created

if (clip != null) {

clip.start();

}

\_bot = new GomokuBot(boardPanel);

nextTurn();

}

public static void updateButton() {

playerButton.setText((Main.currentPlayer == Player.BLACK ? "White " : "Black ") + "Is Playing");

}

private void restartGame() {

boardPanel.clearBoard();

Main.togglePlayer();

}

@Override

public void paint(Graphics g) {

super.paint(g);

g.setColor(Color.white);

g.drawString("Time Played: " + timeCount + " seconds", 50, 50);

g.drawString("Your Turn End In: " + turnTimeCount + " seconds", 650, 50);

}

@Override

public void actionPerformed(ActionEvent e) {

timeCount++;

repaint();

}

public static void turnTimerRestart() {

turnTimeCount = 30;

}

public static void endGame(JFrame frame) {

frame.dispose();

turnTimer.stop();

clip.stop();

clip.close();

if (Main.GAME\_STATUS == gameLogic.GameStatus.black\_player\_win) {

ImageIcon icon = new ImageIcon(

"C:\\Users\\97250\\OneDrive - ORT365Schools\\YodGimal\\Project\\Project\\Img\\black.png");

Image img = icon.getImage();

Image scaledImage = img.getScaledInstance(60, 60, Image.SCALE\_SMOOTH);

ImageIcon scaledIcon = new ImageIcon(scaledImage);

JOptionPane.showMessageDialog(null, "Black Player Wins!", "Game Ended", JOptionPane.INFORMATION\_MESSAGE,

scaledIcon);

} else if (Main.GAME\_STATUS == gameLogic.GameStatus.white\_player\_win) {

ImageIcon icon = new ImageIcon(

"C:\\Users\\97250\\OneDrive - ORT365Schools\\YodGimal\\Project\\Project\\Img\\white.png");

Image img = icon.getImage();

Image scaledImage = img.getScaledInstance(50, 50, Image.SCALE\_SMOOTH);

ImageIcon scaledIcon = new ImageIcon(scaledImage);

JOptionPane.showMessageDialog(null, "White Player Wins!", "Game Ended", JOptionPane.INFORMATION\_MESSAGE,

scaledIcon);

} else if (Main.GAME\_STATUS == gameLogic.GameStatus.draw) {

ImageIcon icon = new ImageIcon(

"C:\\Users\\97250\\OneDrive - ORT365Schools\\YodGimal\\Project\\Project\\Img\\draw.png");

Image img = icon.getImage();

Image scaledImage = img.getScaledInstance(60, 60, Image.SCALE\_SMOOTH);

ImageIcon scaledIcon = new ImageIcon(scaledImage);

JOptionPane.showMessageDialog(null, "It's A Draw!", "Game Ended", JOptionPane.INFORMATION\_MESSAGE,

scaledIcon);

}

}

public static void endGameWrapper() {

endGame(\_frame);

}

public Square get\_currSquare() {

return \_currSquare;

}

public void set\_currSquare(Square \_currSquare) {

this.\_currSquare = \_currSquare;

}

public GameManager get\_Manager() {

return \_Manager;

}

public void set\_Manager(GameManager \_Manager) {

this.\_Manager = \_Manager;

}

public JTextArea get\_log() {

return \_log;

}

public void set\_log(JTextArea \_log) {

this.\_log = \_log;

}

public void removeLastLogLine() {

JTextArea logTextArea = \_log;

int lastNewLineIndex = logTextArea.getText().lastIndexOf("\n");

if (lastNewLineIndex >= 0) {

logTextArea.replaceRange("", lastNewLineIndex, logTextArea.getText().length());

}

}

public BoardPanel getBoardPanel() {

return boardPanel;

}

public void setBoardPanel(BoardPanel boardPanel) {

this.boardPanel = boardPanel;

}

public Settings get\_settings() {

return \_settings;

}

public void set\_settings(Settings \_settings) {

this.\_settings = \_settings;

}

public boolean get\_turn() {

return \_turn;

}

public void set\_turn(boolean turn) {

\_turn = turn;

}

public void nextTurn() {

//\_turn = !\_turn;

if (\_turn) {

timer = new Timer(1000, new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

\_Manager.makeMove();

\_turn = !\_turn;

timer.stop();

}

});

timer.setRepeats(false);

timer.start();

}

else

\_turn = !\_turn;

}

}

##### Main

package gameUI;

public class Main {

public static gameLogic.*GameStatus* *GAME\_STATUS* = gameLogic.*GameStatus*.***in\_progress***;

public static *Player* *currentPlayer* = *Player*.***WHITE***;

public static *Player* togglePlayer() {

*currentPlayer* = (*currentPlayer* == *Player*.***BLACK***) ? *Player*.***WHITE*** : *Player*.***BLACK***;

OuterFrame.*updateButton*();

OuterFrame.*turnTimerRestart*();

return *currentPlayer*;

}

public static void main(String[] args) {

StartPageFrame startPage = new StartPageFrame(new Settings());

EndPageFrame e = new EndPageFrame();

}

}

##### Player

package gameUI;

public enum *Player* {

***BLACK***("B"),

***WHITE***("W"),

***EMPTY***("E");

private final String symbol;

private Player(String symbol) {

this.symbol = symbol;

}

public String getSymbol() {

return symbol;

}

}

##### RulesFrame

package gameUI;

import java.awt.BorderLayout;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.Graphics;

import java.awt.Image;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JEditorPane;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JScrollPane;

public class RulesFrame extends JFrame {

private JButton back;

private Settings \_settings;

public RulesFrame(Settings s) {

super("Gomoku Rules");

\_settings = s;

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

// Create a JEditorPane to display HTML content

JEditorPane editorPane = new JEditorPane();

editorPane.setContentType("text/html");

editorPane.setEditable(false);

// Load the HTML content

String htmlContent = "<!DOCTYPE html>"+

"<html lang=\"en\">"+

"<head>"+

" <meta charset=\"UTF-8\">"+

" <meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">"+

" <title>Gomoku Rules</title>"+

" <style>"+

" body {"+

" font-family: Arial, sans-serif;"+

" margin: 20px;"+

" background-image: url('/img/image.jpg');"+

" }"+

""+

" h1 {"+

" color: #333;"+

" }"+

""+

" p {"+

" line-height: 1.6;"+

" color: #555;"+

" }"+

" </style>"+

"</head>"+

"<body>"+

""+

" <h1>Gomoku Rules</h1>"+

""+

" <p>Gomoku, also known as Five in a Row, is a classic board game played on a grid typically with a 15x15 board. The game is usually played by two players, one using black stones/pieces, and the other using white stones/pieces. The objective of Gomoku is to be the first player to achieve an unbroken row of five stones horizontally, vertically, or diagonally on the game board.</p>"+

""+

" <h2>Basic Rules:</h2>"+

" <ol>"+

" <li>Game Setup:"+

" <ul>"+

" <li>The game is played on a square board with a grid of intersections. A 15x15 board is the most common, but other sizes can be used.</li>"+

" <li>Two players take turns placing their stones on the intersections of the board.</li>"+

" </ul>"+

" </li>"+

" <li>Gameplay:"+

" <ul>"+

" <li>Players take turns placing one of their stones on an empty intersection.</li>"+

" <li>The player using black stones typically makes the first move.</li>"+

" </ul>"+

" </li>"+

" <li>Objective:"+

" <ul>"+

" <li>The primary goal is to create an unbroken row of five stones of your color either horizontally, vertically, or diagonally.</li>"+

" </ul>"+

" </li>"+

" <li>Winning:"+

" <ul>"+

" <li>The first player to achieve an uninterrupted row of five stones wins the game.</li>"+

" <li>The row of five can be vertical, horizontal, or diagonal.</li>"+

" </ul>"+

" </li>"+

" <li>Draw (Tie):"+

" <ul>"+

" <li>If the entire board is filled, and neither player has achieved a row of five, the game is a draw.</li>"+

" </ul>"+

" </li>"+

" </ol>"+

""+

"</body>"+

"</html>";

editorPane.setText(htmlContent);

// Add the JEditorPane to a JScrollPane for scrolling if needed

JScrollPane scrollPane = new JScrollPane(editorPane);

scrollPane.getVerticalScrollBar().setUnitIncrement(5);

// Create a button to go back to the start page

back = new JButton("Back to Start");

back.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

StartPageFrame st = new StartPageFrame(\_settings);

dispose();

}

});

// Add scroll pane and back button to the background label

add(scrollPane, BorderLayout.CENTER);

add(back, BorderLayout.SOUTH);

// Set frame properties

setSize(new Dimension(800, 600));

setLocationRelativeTo(null); // Center the frame on the screen

setVisible(true);

}

}

##### SettingsPage

package gameUI;

import javax.imageio.ImageIO;

import javax.swing.\*;

import javax.swing.border.EmptyBorder;

import javax.swing.event.ChangeEvent;

import javax.swing.event.ChangeListener;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

public class SettingsPage extends JFrame {

private Settings settings;

private BufferedImage \_bg;

public SettingsPage(Settings settings) {

super("Game Settings");

this.settings = settings != null ? settings : new Settings();

JPanel background = new JPanel(new FlowLayout()) {

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

if (\_bg != null) {

// Draw the background image if it's not null

g.drawImage(\_bg, 0, 0, getWidth(), getHeight(), this);

}

}

};

setContentPane(background);

JPanel panel = new JPanel(new GridBagLayout());

panel.setBorder(new EmptyBorder(20, 20, 20, 20));

GridBagConstraints gbc = new GridBagConstraints();

gbc.gridx = 0;

gbc.gridy = 0;

gbc.gridwidth = 2;

gbc.fill = GridBagConstraints.HORIZONTAL;

gbc.insets = new Insets(0, 0, 10, 0);

Font labelFont = new Font("Arial", Font.BOLD, 14);

Font buttonFont = new Font("Arial", Font.PLAIN, 14);

// Game Mode

JLabel gameModeLabel = new JLabel("Game Mode:");

gameModeLabel.setFont(labelFont);

panel.add(gameModeLabel, gbc);

gbc.gridy++;

ButtonGroup gameMode = new ButtonGroup();

JRadioButton pvpButton = createRadioButton("Player vs Player", "PvP");

JRadioButton pvcButton = createRadioButton("Player vs Computer", "PvC");

pvcButton.setSelected(true);

JPanel gameModePanel = new JPanel(new GridLayout(1, 2, 10, 0));

gameModePanel.add(pvpButton);

gameModePanel.add(pvcButton);

panel.add(gameModePanel, gbc);

gameMode.add(pvcButton);

gameMode.add(pvpButton);

// Music Volume

gbc.gridy++;

JLabel musicVolumeLabel = new JLabel("Music Volume:");

musicVolumeLabel.setFont(labelFont);

panel.add(musicVolumeLabel, gbc);

gbc.gridy++;

JSlider musicVolumeSlider = new JSlider(JSlider.HORIZONTAL, 0, 100, settings.getMusicVolume());

musicVolumeSlider.setMajorTickSpacing(10);

musicVolumeSlider.setPaintTicks(true);

musicVolumeSlider.setPaintLabels(true);

musicVolumeSlider.addChangeListener(new ChangeListener() {

@Override

public void stateChanged(ChangeEvent e) {

settings.setMusicVolume(musicVolumeSlider.getValue());

}

});

panel.add(musicVolumeSlider, gbc);

// Player Color

gbc.gridy++;

JLabel playerColorLabel = new JLabel("Player Color:");

playerColorLabel.setFont(labelFont);

panel.add(playerColorLabel, gbc);

ButtonGroup color = new ButtonGroup();

gbc.gridy++;

JRadioButton blackButton = createRadioButton("Black", "Black");

JRadioButton whiteButton = createRadioButton("White", "White");

JPanel playerColorPanel = new JPanel(new GridLayout(1, 2, 10, 0));

playerColorPanel.add(blackButton);

playerColorPanel.add(whiteButton);

panel.add(playerColorPanel, gbc);

color.add(blackButton);

color.add(whiteButton);

blackButton.setSelected(true);

// Difficulty

gbc.gridy++;

JLabel difficultyLabel = new JLabel("Difficulty:");

difficultyLabel.setFont(labelFont);

panel.add(difficultyLabel, gbc);

gbc.gridy++;

JComboBox<String> difficultyComboBox = new JComboBox<>(new String[]{"Easy", "Medium", "Hard"});

difficultyComboBox.setSelectedItem(settings.getDifficulty());

difficultyComboBox.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

settings.setDifficulty((String) difficultyComboBox.getSelectedItem());

}

});

panel.add(difficultyComboBox, gbc);

// Player Name

gbc.gridy++;

JLabel nameLabel = new JLabel("Player Name:");

nameLabel.setFont(labelFont);

panel.add(nameLabel, gbc);

gbc.gridy++;

JTextField nameTextField = new JTextField(settings.getPlayerName());

nameTextField.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

settings.setPlayerName(nameTextField.getText());

}

});

panel.add(nameTextField, gbc);

// Save Button

gbc.gridy++;

gbc.gridwidth = 1;

gbc.fill = GridBagConstraints.NONE;

gbc.anchor = GridBagConstraints.CENTER;

JButton saveButton = new JButton("Save");

saveButton.setFont(buttonFont);

saveButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

settings.setGameMode(gameMode.getSelection().getActionCommand());

settings.setPlayerColor(color.getSelection().getActionCommand());

settings.setPlayerName(nameTextField.getText());

saveSettings();

}

});

panel.add(saveButton, gbc);

background.add(panel);

//add(background);

setSize(500, 500);

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

setLocationRelativeTo(null);

}

private JRadioButton createRadioButton(String text, String actionCommand) {

JRadioButton radioButton = new JRadioButton(text);

radioButton.setFont(new Font("Arial", Font.PLAIN, 14));

radioButton.setForeground(Color.BLACK);

radioButton.setActionCommand(actionCommand);

radioButton.setSelected(actionCommand.equals(settings.getGameMode()));

// Customize radio button appearance

radioButton.setOpaque(false);

radioButton.setBorderPainted(false);

radioButton.setFocusPainted(false);

return radioButton;

}

private void saveSettings() {

dispose();

System.out.println(this.settings.toString());

StartPageFrame spf = new StartPageFrame(settings);

}

}

##### StartPageFrame

package gameUI;

import javax.imageio.ImageIO;

import javax.swing.\*;

import javax.swing.border.EmptyBorder;

import javax.swing.plaf.basic.BasicButtonUI;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

public class StartPageFrame extends JFrame {

private JLabel \_imageLabel;

private BufferedImage \_bg;

private JLabel \_textLabel;

private JButton \_startButton;

private JButton \_rulesButton;

private JButton \_settingsButton;

private Settings \_settings;

public StartPageFrame(Settings settings) {

super("Start Page Gomoku");

\_settings = settings;

try {

\_bg = ImageIO.read(new File("C:\\Users\\liroy\\Desktop\\Project3.0\\img\\pic2.jpg"));

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

// Set the content pane to be a JPanel with BorderLayout

JPanel buttonsPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10000, 20)) {

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

if (\_bg != null) {

// Draw the background image if it's not null

g.drawImage(\_bg, 0, 0, getWidth(), getHeight(), this);

}

}

};

setContentPane(buttonsPanel);

\_textLabel = new JLabel("\tWelcome to Gomoku!");

\_textLabel.setFont(new Font("Arial", Font.BOLD, 20));

\_textLabel.setForeground(Color.black);

\_textLabel.setHorizontalAlignment(SwingConstants.CENTER);

\_textLabel.setBounds(200, 50, 400, 50);

JLabel name = new JLabel("Hello " + \_settings.getPlayerName());

Font font1 = new Font("SansSerif", Font.BOLD, 20);

name.setFont(font1);

name.setOpaque(false);

name.setForeground(Color.black);

name.setBounds(300, 100, 200, 50);

buttonsPanel.add(name);

\_rulesButton = new JButton("Rules");

styleButton(\_rulesButton);

\_rulesButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

RulesFrame rFrame = new RulesFrame(\_settings);

dispose();

}

});

\_startButton = new JButton("Start Game");

styleButton(\_startButton);

\_startButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

LoadingScreen loading = new LoadingScreen(\_settings);

dispose();

}

});

\_settingsButton = new JButton("Settings");

styleButton(\_settingsButton);

\_settingsButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

openSettingsPage();

}

});

add(\_textLabel, BorderLayout.NORTH);

buttonsPanel.add(\_rulesButton);

buttonsPanel.add(\_startButton);

buttonsPanel.add(\_settingsButton);

// Set frame properties

setSize(800, 400); // Adjusted width to accommodate the image

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

setVisible(true);

}

private void styleButton(JButton button) {

button.setFont(new Font("Arial", Font.PLAIN, 16));

button.setPreferredSize(new Dimension(300,50));

button.setBackground(new Color(0, 153, 51));

button.setForeground(Color.WHITE);

button.setFocusPainted(false);

button.setBorderPainted(false);

button.setUI(new StyledButtonUI());

}

private void openSettingsPage() {

SettingsPage settingsPage = new SettingsPage(new Settings());

settingsPage.setVisible(true);

dispose();

}

public Settings get\_settings() {

return \_settings;

}

public void set\_settings(Settings \_settings) {

this.\_settings = \_settings;

}

}

class StyledButtonUI extends BasicButtonUI {

@Override

public void installUI(JComponent c) {

super.installUI(c);

AbstractButton button = (AbstractButton) c;

button.setOpaque(false);

button.setBorder(new EmptyBorder(5, 15, 5, 15));

}

@Override

public void paint(Graphics g, JComponent c) {

AbstractButton b = (AbstractButton) c;

paintBackground(g, b, b.getModel().isPressed() ? 2 : 0);

super.paint(g, c);

}

private void paintBackground(Graphics g, JComponent c, int yOffset) {

Dimension size = c.getSize();

Graphics2D g2 = (Graphics2D) g;

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

g.setColor(c.getBackground().darker());

g.fillRoundRect(0, yOffset, size.width, size.height - yOffset, 10, 10);

g.setColor(c.getBackground());

g.fillRoundRect(0, yOffset, size.width, size.height + yOffset - 5, 10, 10);

}

@Override

protected void paintButtonPressed(Graphics g, AbstractButton b) {

paintBackground(g, b, 2);

}

@Override

protected void paintText(Graphics g, JComponent c, Rectangle textRect, String text) {

AbstractButton b = (AbstractButton) c;

ButtonModel model = b.getModel();

if (model.isPressed()) {

textRect.y += 2;

}

super.paintText(g, c, textRect, text);

}

@Override

protected void installListeners(AbstractButton b) {

super.installListeners(b);

b.addMouseListener(new HoverListener());

}

private static class HoverListener extends MouseAdapter {

@Override

public void mouseEntered(MouseEvent e) {

AbstractButton button = (AbstractButton) e.getSource();

button.setBackground(Color.red); // Change background color on hover

}

@Override

public void mouseExited(MouseEvent e) {

AbstractButton button = (AbstractButton) e.getSource();

button.setBackground(new Color(0, 153, 51)); // Reset background color on exit

}

}

}

##### Settings

package gameUI;

public class Settings {

private String gameMode; // PvP or PvC

private int musicVolume;

private String playerColor; // Black or White

private String difficulty;

private String playerName;

public Settings(String gameMode, int musicVolume, String playerColor, String difficulty, String playerName) {

super();

this.gameMode = gameMode;

this.musicVolume = musicVolume;

this.playerColor = playerColor;

this.difficulty = difficulty;

this.playerName = playerName;

}

public Settings() {

this.gameMode = "PVC";

this.musicVolume = 30;

this.playerColor = "Black";

this.difficulty = "Easy";

this.playerName = " ";

}

public String getGameMode() {

return gameMode;

}

public void setGameMode(String gameMode) {

this.gameMode = gameMode;

}

public int getMusicVolume() {

return musicVolume;

}

public void setMusicVolume(int musicVolume) {

this.musicVolume = musicVolume;

}

public String getPlayerColor() {

return playerColor;

}

public void setPlayerColor(String playerColor) {

this.playerColor = playerColor;

}

public String getDifficulty() {

return difficulty;

}

public void setDifficulty(String difficulty) {

this.difficulty = difficulty;

}

public String getPlayerName() {

return playerName;

}

public void setPlayerName(String playerName) {

this.playerName = playerName;

}

*@Override*

public String toString() {

return "Settings [gameMode=" + gameMode + ", musicVolume=" + musicVolume + ", playerColor=" + playerColor

+ ", difficulty=" + difficulty + ", playerName=" + playerName + "]";

}

}

##### Square

package gameUI;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Graphics;

import java.awt.Image;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import javax.swing.JPanel;

import javax.swing.JTextArea;

import javax.swing.JTextField;

public class Square extends JPanel {

private Player player;

private boolean hasSoldier;

private int \_i, \_j;

private OuterFrame oFrame;

private Square \_currSquare;

public Square(int i, int j) {

super();

\_i = i;

\_j = j;

\_currSquare = this;

player = Player.EMPTY;

Color c = new Color(154, 115, 26);

setBackground(c);

hasSoldier = false;

setPreferredSize(new Dimension(30, 30));

addMouseListener(new MouseAdapter() {

@Override

public void mousePressed(MouseEvent e) {

super.mousePressed(e);

placePlayer();

}

});

}

public void placePlayer() {

if (!oFrame.get\_turn()) {

if (!hasSoldier) {

System.out.println(oFrame.get\_turn()?"bot":"player");

hasSoldier = true;

player = Main.togglePlayer();

oFrame.set\_currSquare(\_currSquare);

oFrame.get\_log().append(Main.currentPlayer + " player has put player in:" + \_currSquare.get\_i() + ","

+ \_currSquare.get\_j() + "\n");

oFrame.get\_Manager().startGame(\_currSquare.get\_i(), \_currSquare.get\_j());

repaint();

oFrame.nextTurn();

}

}

else {

if (!hasSoldier) {

System.out.println(oFrame.get\_turn()?"bot":"player");

hasSoldier = true;

player = Main.togglePlayer();

oFrame.set\_currSquare(\_currSquare);

oFrame.get\_log().append(Main.currentPlayer + " Bot has put player in:" + \_currSquare.get\_i() + ","

+ \_currSquare.get\_j() + "\n");

oFrame.get\_Manager().startGame(\_currSquare.get\_i(), \_currSquare.get\_j());

repaint();

oFrame.nextTurn();

}

}

}

public void botPlacePlayer(int row, int col) {

if (!hasSoldier) {

System.out.println(oFrame.get\_turn()?"bot":"player");

hasSoldier = true;

player = Main.togglePlayer();

oFrame.set\_currSquare(\_currSquare);

oFrame.get\_log().append(Main.currentPlayer + " Bot has put player in:" + \_currSquare.get\_i() + ","

+ \_currSquare.get\_j() + "\n");

oFrame.get\_Manager().startGame(\_currSquare.get\_i(), \_currSquare.get\_j());

repaint();

oFrame.nextTurn();

}

}

public void paintComponent(Graphics g) {

super.paintComponent(g);

g.drawLine(0, 20, 40, 20);

g.drawLine(20, 0, 20, 40);

if (hasSoldier) {

g.setColor(player == Player.BLACK ? Color.black : Color.white);

g.fillOval(10, 10, 20, 20);

}

}

public void clearSquare() {

hasSoldier = false;

player = null;

repaint();

}

public int get\_i() {

return \_i;

}

public void set\_i(int \_i) {

this.\_i = \_i;

}

public int get\_j() {

return \_j;

}

public void set\_j(int \_j) {

this.\_j = \_j;

}

public OuterFrame getoFrame() {

return oFrame;

}

public void setoFrame(OuterFrame oFrame) {

this.oFrame = oFrame;

}

public boolean isHasSoldier() {

return hasSoldier;

}

public void setHasSoldier(boolean hasSoldier) {

this.hasSoldier = hasSoldier;

}

public Player getPlayer() {

return player;

}

public void setPlayer(Player player) {

this.player = player;

}

}

# רפלקציה

בפרויקט המרתק של GOUMKO ב-Java, התמודדתי עם אתגרים טכניים וארגוניים מרתקים. העבודה על המשחק הייתה חוויה מעוררת השראה, שהשאירה לי מעמקי לימוד חדשים והיכולת ליישם אותם בפועל. במהלך הפרויקט, התקדמתי לתוך עולם הפיתוח המשוכלל יותר, והתמודדתי עם שאלות מרתקות כיצד לשפר את תהליך הפיתוח, את איכות הקוד ואת חווית המשתמש.

במהלך העבודה, חשפתי את עצמי לשאלות איכותיות כיצד לשפר את חווית המשתמש, תוך התחשבות בצרכי הלקוחות האפשריים ובכלי נוספים שיכולים להגדיל את התחרותיות של המשחק. התפתחות זו הובילה אותי להתייחסות מעמיקה יותר לתהליך הפיתוח, באמצעות תכנון מקדים והגדרת דרישות מפורטות, תוך שימת דגש על ניהול זמן ותקשורת יעילה עם חברי הצוות.

מסקנתי העיקרית מפרויקט זה היא שקיימות תהליכי פיתוח מתודולוגיים וכלים טכנולוגיים שיכולים לשפר באופן משמעותי את איכות הפרויקט ואת חווית המשתמש. הייתי מעוניינת להתמקד יותר בתהליכים אלו כבר בהתחלה, ולהשקיע מאמץ נוסף בתכנון ובהתאמת הפתרונות לצרכי המשתמש באופן מדויק יותר.

באופן כללי, הפרויקט חידש את תפיסתי לגבי התפתחות תוכנה ועבודת צוות, והעניק לי יכולת לעבוד באופן יצירתי ומקצועי בתחום פיתוח התוכנה.

# בביבליוגרפיה

* youtube
* GeeksforGeeks
* stack overflow
* javapoint