**עבודת גמר**

**לקבלת תואר טכנאי**

**הנדסת תוכנה**

**מכללה:** אורטחולון

**שם הפרויקט:** קורידור

**שם הסטודנט:** רני מורה

**ת"ז:** 325106896

**שם המנחה:** הילה ויינשטיין

**מאי 2021 תשפ"א**

**תוכן עניינים**

[**1. תקציר** 3](#_Toc70812213)

[**2. תיאור הנושא** 3](#_Toc70812214)

[**3. רקע תיאורטי** 4](#_Toc70812218)

[**4. תיאור הבעיה האלגוריתמית** 4](#_Toc70812219)

[**5. סקירת האלגוריתמים בתחום הבעיה** 4](#_Toc70812220)

[**6. מושגים** 5](#_Toc70812221)

[**7. אסטרטגיה** 7](#_Toc70812222)

[**8. מבנה נתונים** 12](#_Toc70812223)

[**9. תרשים מחלקות** 13](#_Toc70812224)

[**10. ארכיטקטורה Top-Down Level Design** 14](#_Toc70812225)

[**11. תיאור סביבת העבודה ושפת התכנות** 14](#_Toc70812226)

[**12. תיאור הממשקים** 15](#_Toc70812227)

[**13. אלגוריתם ראשי** 21](#_Toc70812228)

[**14. תיאור המחלקות הראשיות** 25](#_Toc70812229)

[.15**תוכנית ראשית** 52](#_Toc70812230)

[**16. מדריך למשתמש** 54](#_Toc70812231)

[**17. קוד הפרויקט** 58](#_Toc70812232)

[**18. רפלקציה** 180](#_Toc70812233)

## **1. תקציר**

הפרויקט שלי הוא גירסא ממוחשבת של המשחק קורידור. קורידור הוא משחק לוח אסטרטגי משנות ה90. ספר הפרויקט סוקר את הפרויקט שלי ואלו הן הנקודות המרכזיות שהוא כולל:

* תיאור המשחק ורקע תיאורטי
* אלגוריתם- תיאור האלגוריתם, וסקירה בנושא
* מבנה נתונים, תרשים מחלקות וארכיטקטורה
* סביבת העבודה והשפה
* הסבר על הפונקציות והאלגוריתמים בפרויקט
* מדריך למשתמש
* רפלקציה אישית

כמו כן, ניתן למצוא את הקוד במלואו.

## **2. תיאור הנושא**

### 

### **תכולת משחק 1**

המערכת מורכבת מלוח משחק שגודלו 9X9, כאשר כל יש גישה בין כל התאים בתחילת המשחק ועם התקדמותו הגישה בין התאים משתנה לפי המחסומים שהשחקנים מיקמו.  
המשחק מאותחל כך שעבור כל שחקן מסודרים 10 מחסומים שניתנו לו לשים במהלך המשחק בתואם לצבעו של השחקן (שאותו לשחקן יש אפשרות לבחור או יקבע באופן רנדומאלי).  
המבנה של הלוח המאפשר תזוזה אנכית ואופקית, כל עוד התא בסמוך ריק. במידה והתא בכיוון שהשחקן רוצה לזוז מכיל:

מחסום, השחקן לא יוכל להתקדם לכיוון זה ועליו לבחור צעד אחר לתור

יריב, במידה ואין מחסום מאחורי היריב (בכיוון שהשחקן מתכוון לזוז עליו) השחקן מדלג מעל היריב.

במידה וקיים מחסום מאחורי היריב, השחקן רשאי לזוז באלכסון ממקומו(לאחד מצדיו של היריב)

.   
המשחק מנוהל בין שחקנים אנושים או שחקנים ממוחשבים, כאשר ניתן לקבוע את סוג השחקן במסך ההגדרות ליפני תחילת המשחק וניתן להוסיף עד 4 שחקנים (הכמות מינימאלית של שחקנים היא 2).

בכל תור כל אחד מהשחקנים בוחר מהלך באסטרטגיה שהוא בוחר, או לקדם את השחקן שלו או למקם מחסום על לוח המשחק.  
ניתן לבצע בתור רק מהלך אחד מבין השניים בתור של שחקן.

### **2 מטרת משחק**

מטרת המשחק היא להגיע עם השחקן לצד המקביל עליו בלוח ליפני שאר השחקנים(מתחיל בימין להגיע לשמאל, מתיל למעלה להגיע למטה ולהפך) , טרם שאר השחקנים יבצעו זאת.

### **3 סיום משחק**

המשחק מסתיים כאשר אחד השחקנים הצליח להגיע לצד שעליו להגיע.

## **3. רקע תיאורטי**

משחקי לוח ידועים עוד מתקופת מצרים העתיקה בו שיחקו ב[שנת](https://he.wikipedia.org/wiki/%D7%A1%D7%A0%D7%AA) עוד כ-3500 שנים לפנה"ס.

קורידור הוא משחק אסטרטגיה אינטואיטיבי בן 2 או 4 שחקנים שתוכנן על ידי מירקו מרצ'סי ופורסם על ידי Gigamic Games. קורידור קיבל את פרס משחק חשיבה של Mensa Mind בשנת 1997 ואת משחק השנה בארצות הברית, צרפת, קנדה ובלגיה.

המשחק מפתח כושר ריכוז, ניהול משאבים וניתוח מצבים, דורש חשיבה אסטרטגית ומקוריות

## **4. תיאור הבעיה האלגוריתמית**

לדמות יריב- שחקן אנושי

פיתוח רמות שונות של חשיבה למחשב, איך ניתן לגרו ם למחשב להבין מה הפעולה שעליו לעשות כדי שיהיו לו את הסיכויים הטובים ביותר לנצח האם כדאי לו להתקדם או לשים מכשול בדרך ליריב? אם להתקדם אז לאן? ואם לשים מכשול איפה?. בדיקת תקינות פעולה- אסור להציב מכשול שיגרום לשחקן היריב להילכד מכל הכיוונים ללא דרך להגיע לנקודת הסיום

האלגוריתמיקה במשחק באה לידי ביטוי בכמה אופנים, ומתבססת בעיקר על אלגוריתם BFS ומשתמשת בו גם לבדיקות תקינות במשחק וגם במהלכים של היריב.

האופן הראשון שבוא האלגוריתם פועל הוא כאשר שחקן מניח מחסום, על האלגוריתם לבדוק שאכן לכל השחקנים ישנה עדיין דרך להגיע ליעד שלהם לאחר מיקום המחסום, במידה ונימצא כי המיקום אינו תקין המחסום יחזור לשחקן ששם אותו ועליו יהיה לבחור צעד אחר לתורו.

האופן השני הוא לשיקולים של הCPU במשחק – שחקן שאינו אמור להיות תחת שליטה של המשתמש אלא של המחשב.

האלגוריתם בודק מה המסלול הכי קצר של כל השחקנים, מה שעוזר לCPU קודם כל להחליט האם כדאי לו לשים מחסום לפי חישוב של המצב הנוכחי של כל השחקנים לעומת שלו , במידה ונימצא שלא כדאי לו לחסום ליריב את הדרך הוא בודק לאיזה כיוון הוא צריך להתקדם כדי לממש את הדרך הקצרה ביותר עבורו.

## **5. סקירת האלגוריתמים בתחום הבעיה**

במשחק שלי קיים אלגוריתם BFS שפועל על גרף שמציג את לוח המשחק, השימוש באלגוריתם הוא למציאת המסלול שאפשר להגיע איתו ליעד בדרך הקצרה ביותר.

אלגוריתמים אחרים שיכולים לפתור את תחום הבעיה.

DFS- אלגוריתם לחציה או חיפוש בגרף. האלגרויתם מתחיל מהנקודה הנתונה\ השורש ובודקת ככל האפשר לאורך כל שכן לפני מעקב אחורה

Dijkstra-אלגוריתם למציאת מסלול הקצר ביותר בין צמתים בתוך גרף עם המרחק הנמוך ביותר, מחשבת את מרחק דרכו ומעדכנת את מרחק השכן אם הוא קטן יותר.

## **6. מושגים**

במהלך העבודה שלי על הפרויקט למדתי דברים רבים והמון מושגים, למדתי על גרפים ועל השימוש הרחב שלהם בעולם המחשבים ושימוש באלגוריתם חיפוש, מה שמאוד פיתח לי חשיבה והבנה כיצד דברים אחרים בעולם עובדים.

למדתי על עיצוב ואסתטיקה במשחקים, החשיבה האלגוריתמית שלי עם בינה מלאכותית השתפרה והתפתחה(היה לנו גם בכיתה י'ב פרוייקט בו היינו צריכים ליצור בינה מלאכותית למשחק שלנו) .

למדתי המון על THREAT וחוטים והשימוש בזה בהפרדה של פעולות משמעותיות וחילוק ליבת הTHREAT.

תהליכון, thread, הוא קטע קוד מוגדר, אשר יבוצע בנפרד מה"זרימה" הרגילה של התוכנית, בצורה מקבילית. כל תוכנית בג'אווה תרוץ לפחות בתור תהליכון בודד, שיווצר ע"י המכונה הוירטואלית.

למדתי על שימוש בEVENTS ואירועים ואיך לשלב ולגוון אותם במשחק בדרכים רבות.

בהנדסת תוכנה, בתכנות מונחה אירועים (Event-driven programming) הוא תפיסה בתכנות. הרעיון הוא שבתוך תוכנית המחשב קיימים חלקים, הממתינים לקבלת אות. האות נקרא "אירוע" (event) והוא מתקבל כאשר מתרחש אירוע מסוים במערכת, אליו קשוב היישום. האירועים בדרך כלל יהיו פנימיים, בין שני חלקים של אותה תוכנה, אך גם יכולים להיות חיצוניים בין תוכנה אחת לשנייה.

למדתי הרבה על הורשה ופולימורפיזם בתכנות מונחה עצמים ועל השימוש שניתן לבצע איתו ביצירת משחקים

הורשה (תורשה או ירושה- Inheritance) הינה אחד העקרונות הראשיים הבאים לידי ביטוי בתכנות מונחה עצמים (OOP). הורשה מאפשרת לרשת ממחלקה מסויימת את “היכולות” שלה, שהן התכונות והמתודות שלה- איברי המחלקה (Class Members) ע”י מחלקה אחרת שניקראת המחלקה היורשת (Derived-Class/ Sub-Class). המחלקת שממנה ירשו נקראת מחלקת בסיס או מחלקת אב (Base-Class/ SuperClass). בדומה לביולוגיה, ניתן להתייחס למופע מחלקה שהוא אובייקט שלה כאל אובייקט הבן שיש לו את אותן היכולות, הפונקציות והתכונות (attributes) של מחלקת האב (גנים).

פולימורפיזם (רב צורתיות/ רב- טיפוסיות) היא היכולת של אובייקט שנוצר מטיפוס מסויים להיות בו זמנית אובייקט מטיפוסים אחרים, כך שניתן להריץ עליו מתודות של מחלקות שונות. באמצעות מנגנון הפולימורפיזם ניתן להתייחס למספר אובייקטים בעץ תורשה שהם מסוגים שונים בצורה אחידה דרך מאפיין בסיסי המשותף בינהם. כאשר האובייקטים הם מטיפוסים שונים, ניתן להסתכל על המחלקה המשותפת לכולם (למשל מחלקת Object) ולבנות מערך הפניות לאובייקטים, כאשר ההפניות עצמן מהטיפוס הבסיסי ביותר שמשותף לכולם. ניתן לזמן בכל פעם מתודות השייכות למחלקות שונות באמצעות שימוש בהפניה (רפרנס/ מצביע/ reference) מטיפוסים שונים בעץ התורשה שהיא בחוזקים שונים, כלומר לכל אחת יכולות שונות. למעשה הפניה יכולה להיות חלשה יותר או שווה ביכולותיה לאובייקט המוצבע על ידיה. ככל שיורדים בעץ התורשה, כך המחלקות בעלי יותר יכולות ועל כן הן חזקות יותר. לסיכום, תכונת הפולימורפיזם מאפשרת לבנות מבני נתונים (מערכים, רשימות) המכילים הפניות מטיפוס בסיסי זהה המצביעות לאובייקטים מטיפוסים שונים וברמות שונות על אותו עץ תורשה (היררכיית תורשה) ולהריץ פונקציות מתאימות על כל אובייקט לסוגו על ידי המרת ההפניות למטה (DownCasting).

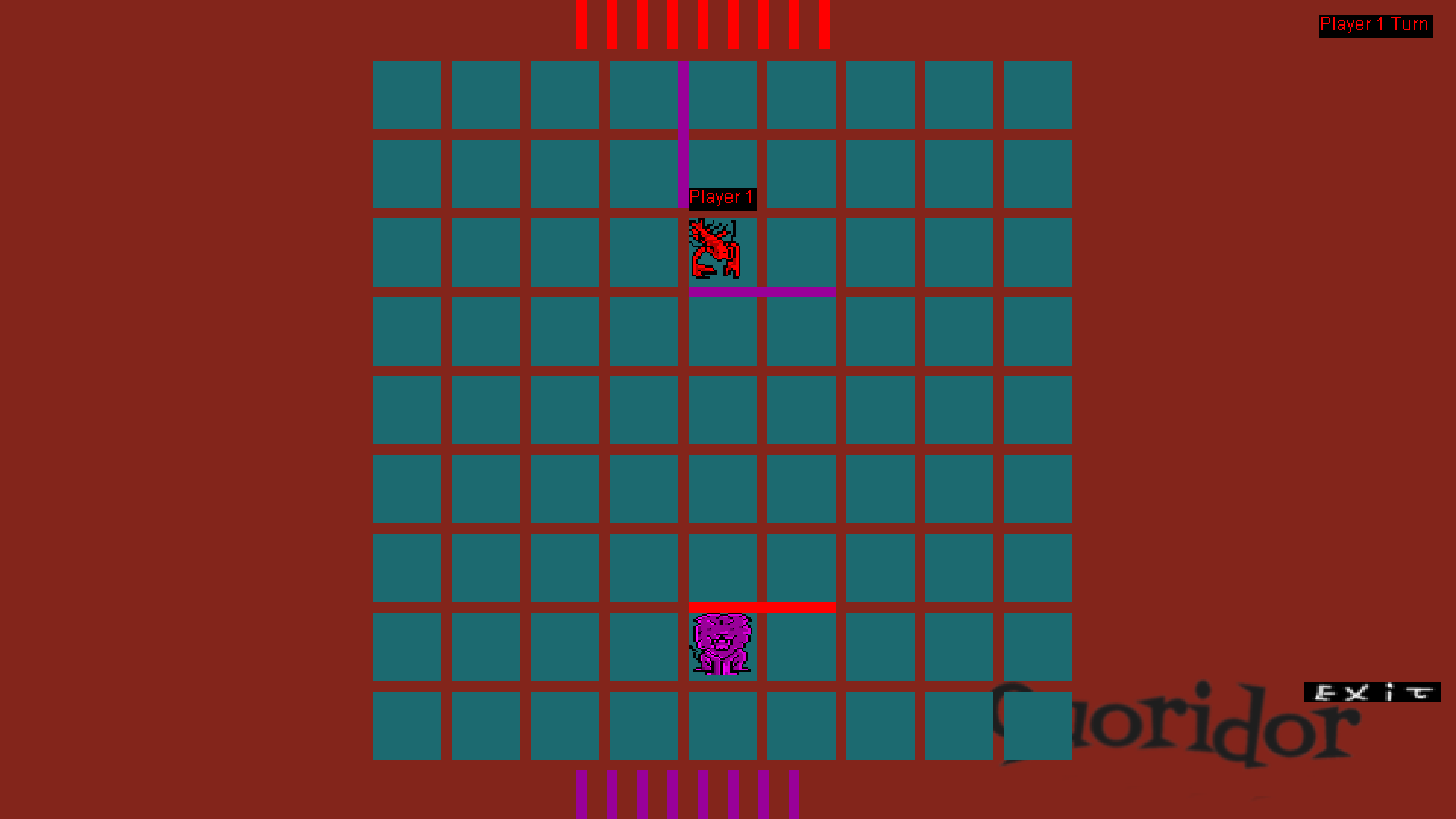
וב-JAVA, כל אובייקט יורש באופן אוטומטי ממחלקת העל הבסיסית Object ומכאן שכל אובייקט בג’אווה הוא בו זמנית מסוג שני טיפוסים לפחות, מטיפוס האובייקט ומטיפוס Object.

לסיכום במהלך השנה למדתי המון מונחים ומושגים בעולם המשחקים והמחשבים

חלקם פשוט דברים שהסתדרו לי בראש כמו איך מנגנון כלשהו עובד במשחק מסוים שאני מכיר, נהנתי מעוד לעבוד על הפרוייקט(ים) במהלך השנה וההנה הזאת הביאה לי השארה לנסות לפתח עוד משחקים בעצמי ולא בשביל פרויקט במסגרת כולשהי.

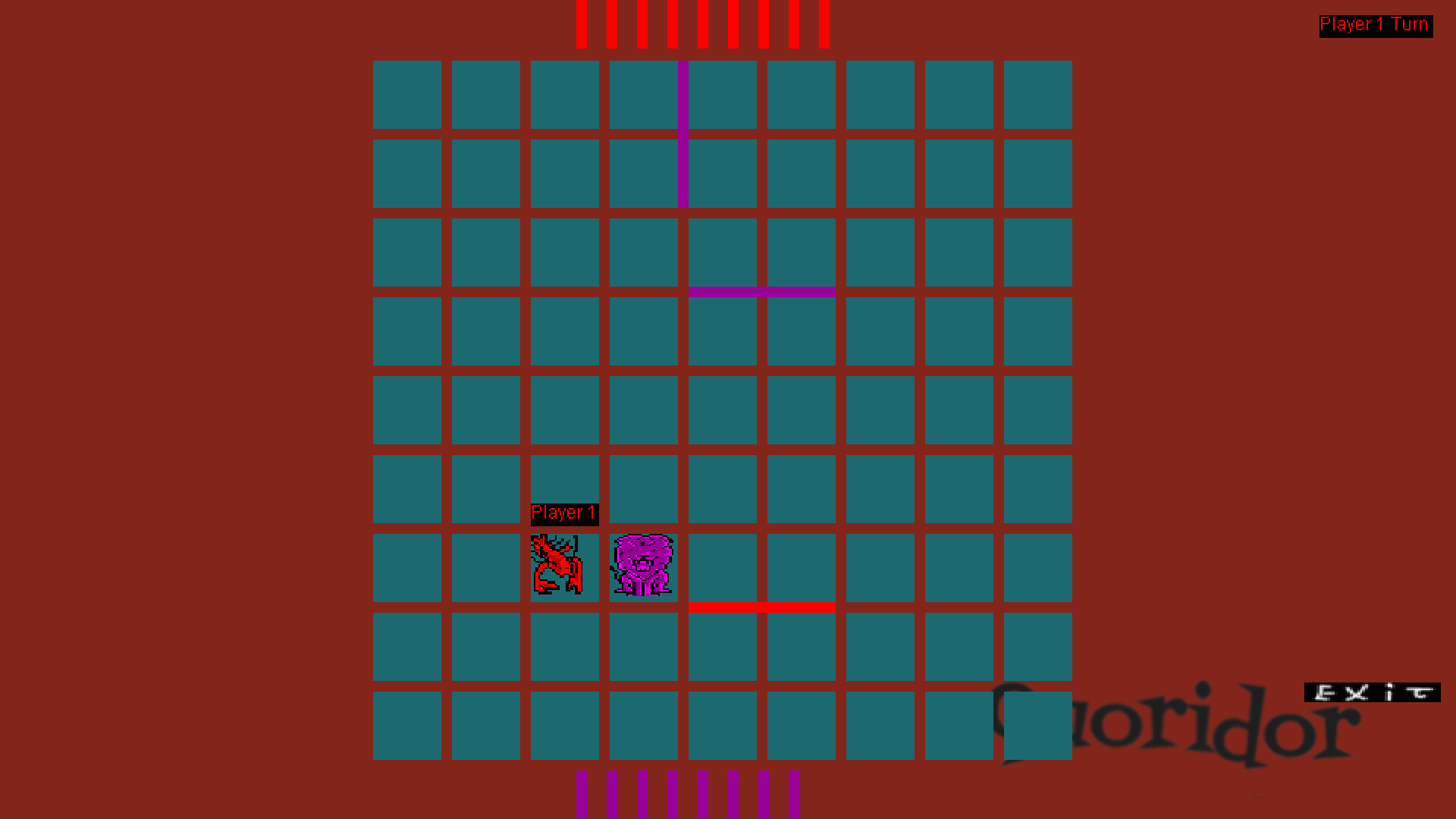
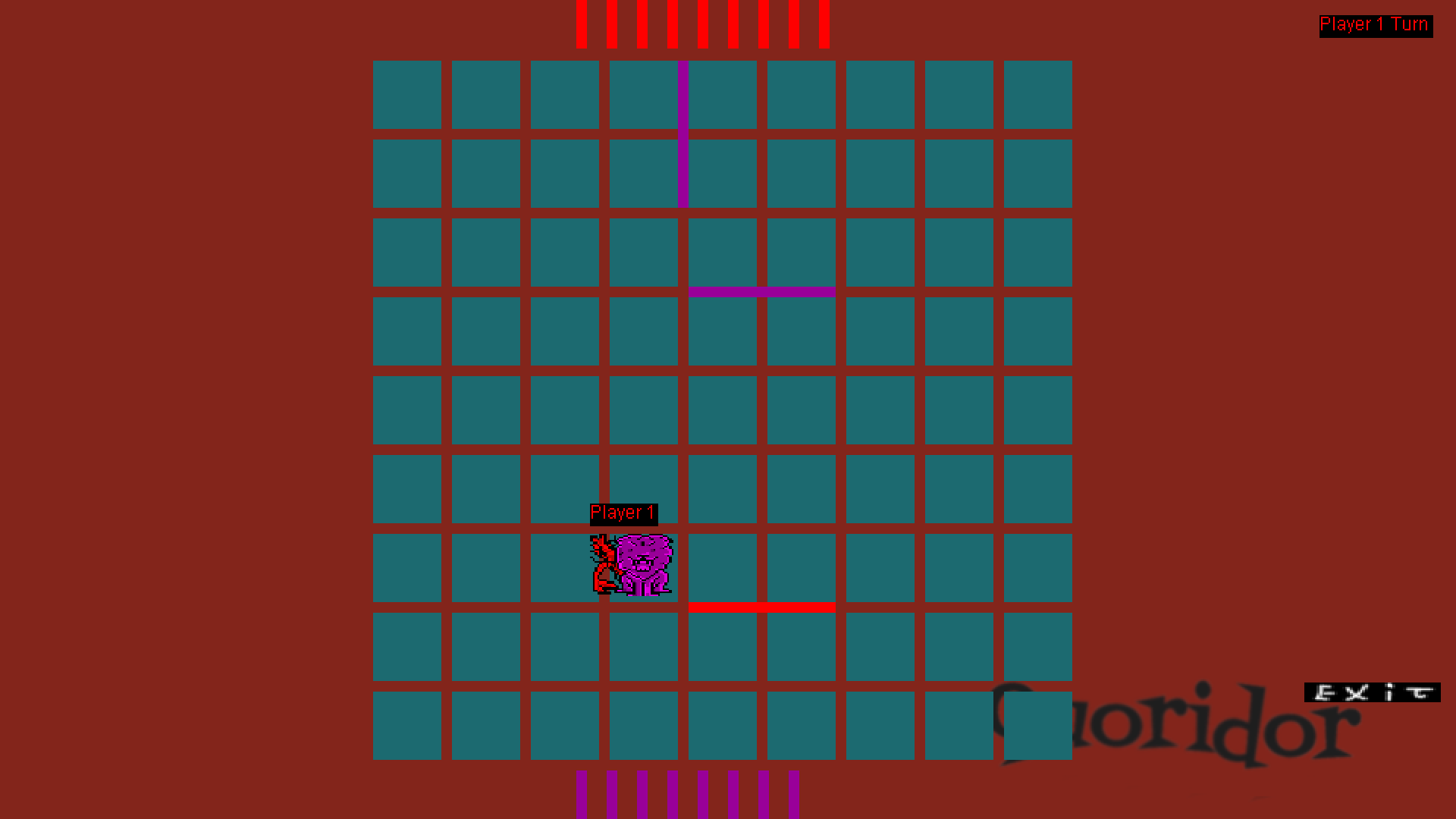
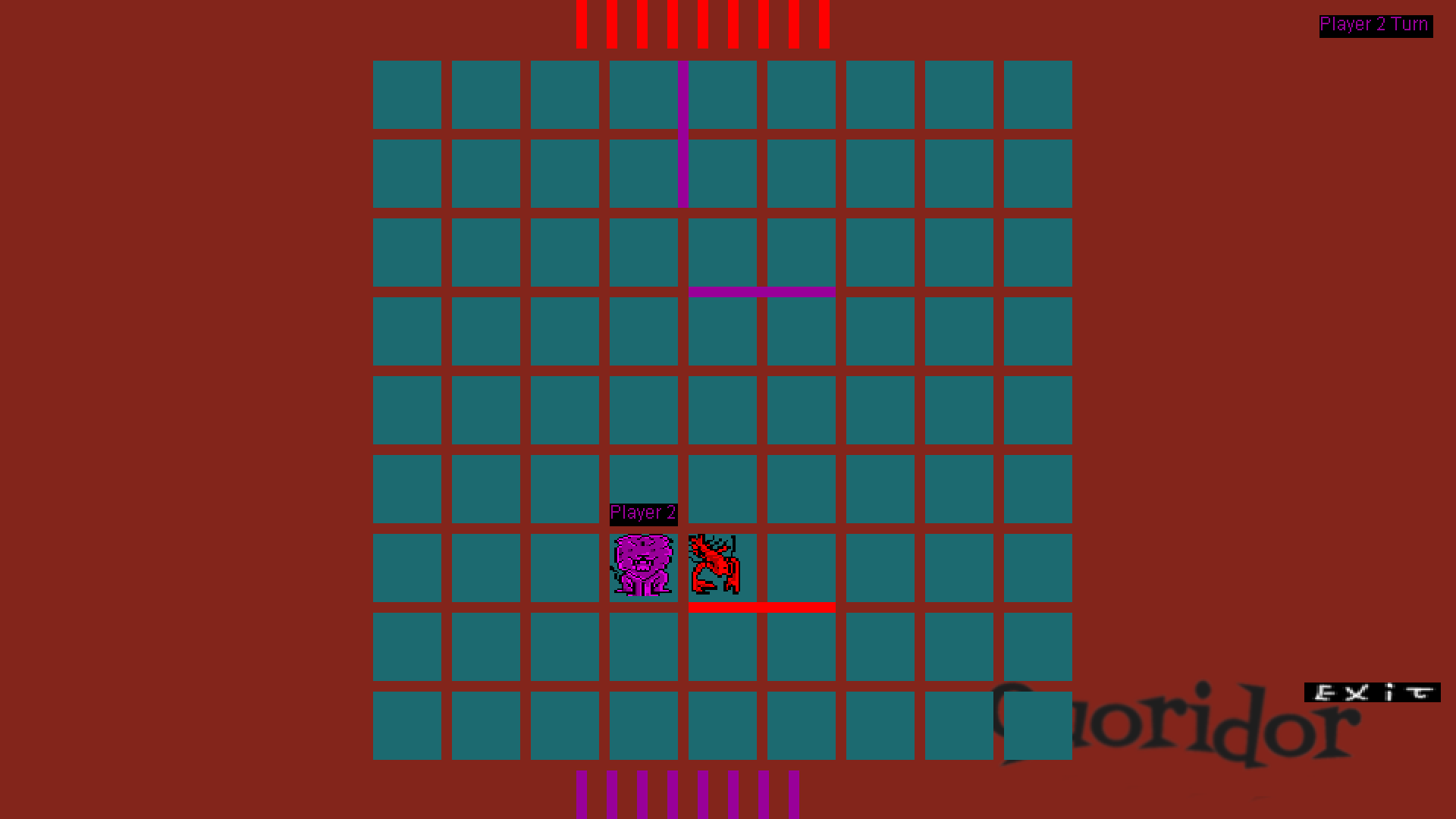
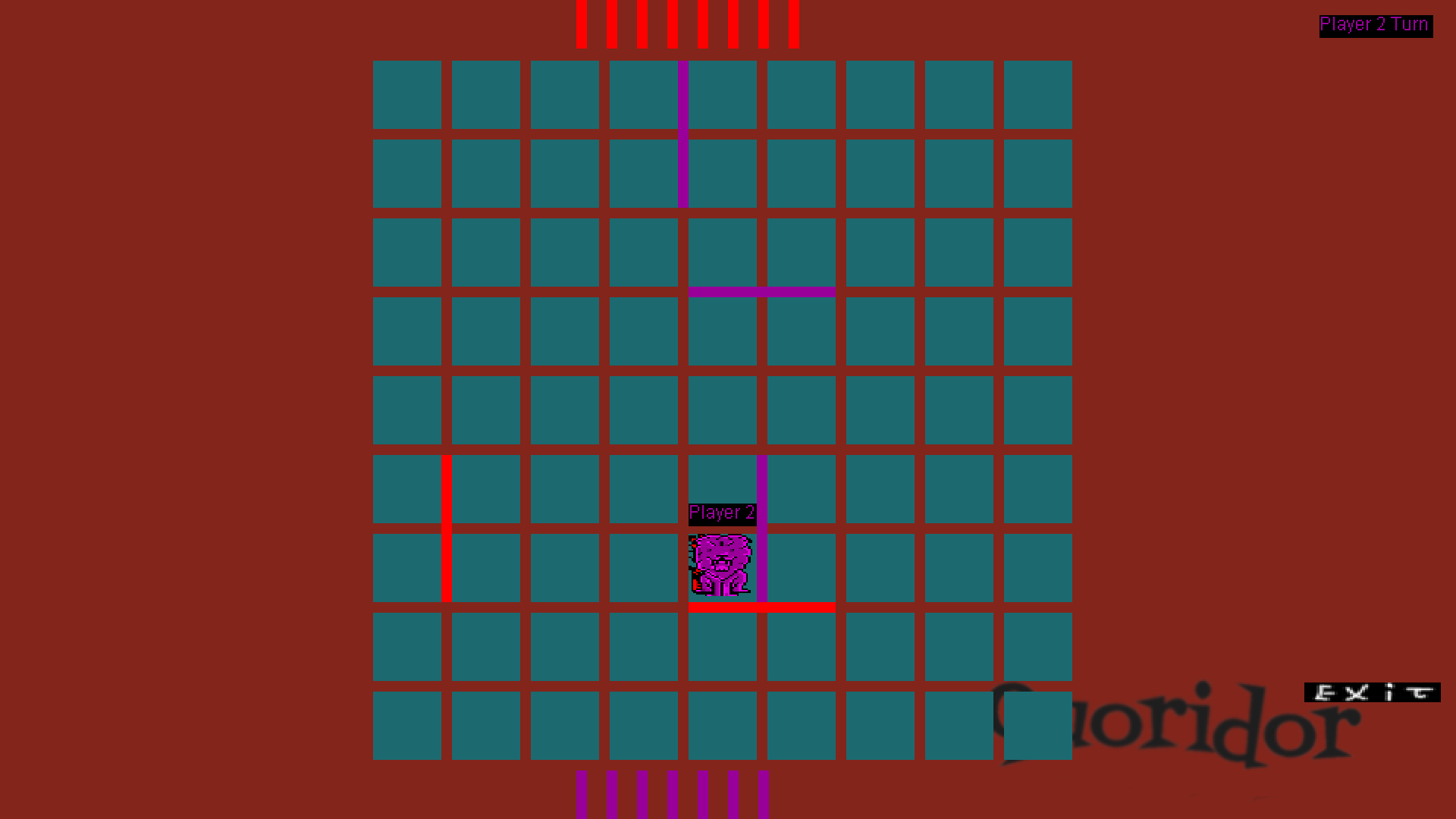
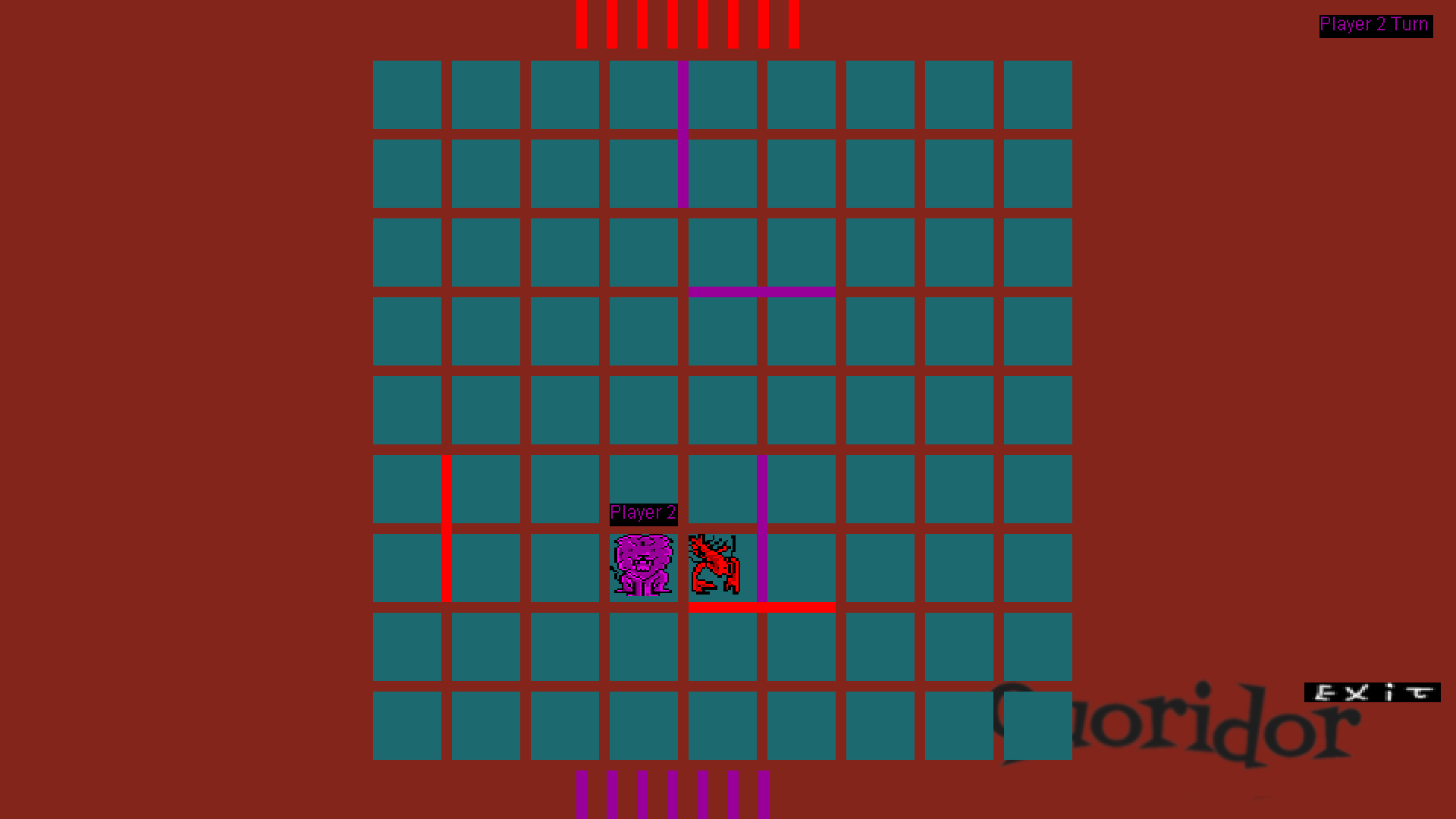
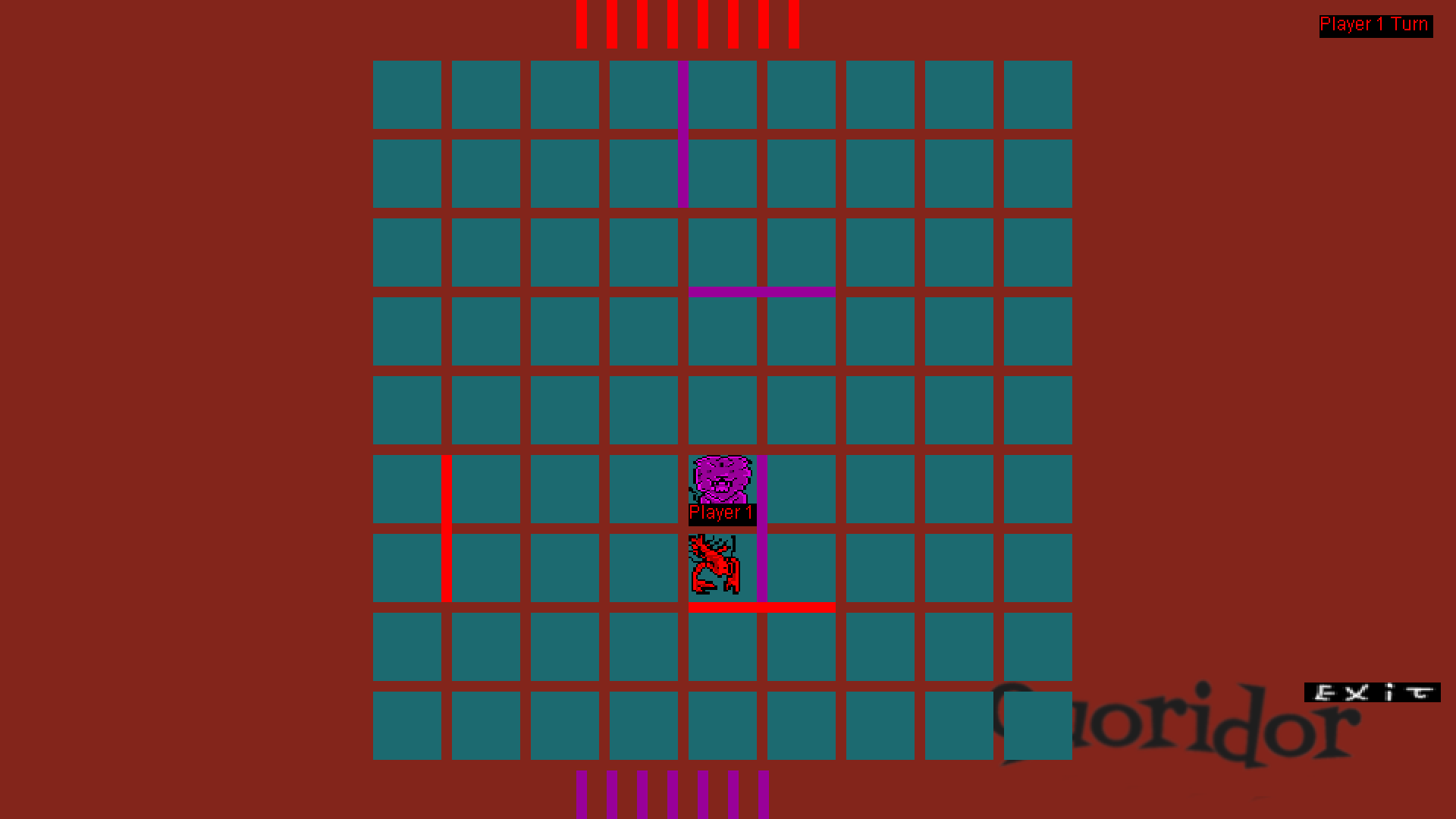
## **7. אסטרטגיה**

אסטרטגיית המשחק היא להגיע עם השחקן לצד המקביל עליו בלוח ליפני שאר השחקנים(מתחיל בימין להגיע לשמאל, מתיל למעלה להגיע למטה ולהפך) , ועליו לעשות כל שיוכל על מנת לעקב את היריבים על מנת להגיע ראשון למטרה שלהם, טרם שאר השחקנים יבצעו זאת, בחרתי אסטרטגיה זו מכיוון שהיא מבוססת על האסטרטגית המשחק האמיתי ואני ניסיתי ללכת כמה שיותר בעקבות חוקי המשחק האמיתי פרט לשינויים קלים שאני מצאתי לנכון לשנות כדי שהמשחק יהיה יותר מעניין לדוגמא – במשחק האמיתי לא ניתן לדלג מעל יותר מיריב אחד אף במשחק שלי אני אפשרתי לדלג מעל כל היריבים במידה והם עומדים סדיר אחד אחרי השני, במשחק האמיתי לא ניתן לשים מחסום בצורת פלוס היכן שישנו מחסום מחובר (מכיוון שמבחינה פיזית לא ניתן לעשות את זה) אך במשחק שלי בחרתי שלא לפסול את המהלך הזה, כי בכל זאת שהוא משחק מחשב ואינו מוגבל ביכולות פיזיות מציאותיות, והדבר יכול לגרום למשחק להגיע למצבים מעניינים יותר שלא היו יכולים להגיע עליהם במשחק האמיתי .פרט לדברים אלו לא התחללו עוד שינויים ב אסטרטגיית המשחק או בחוקים.

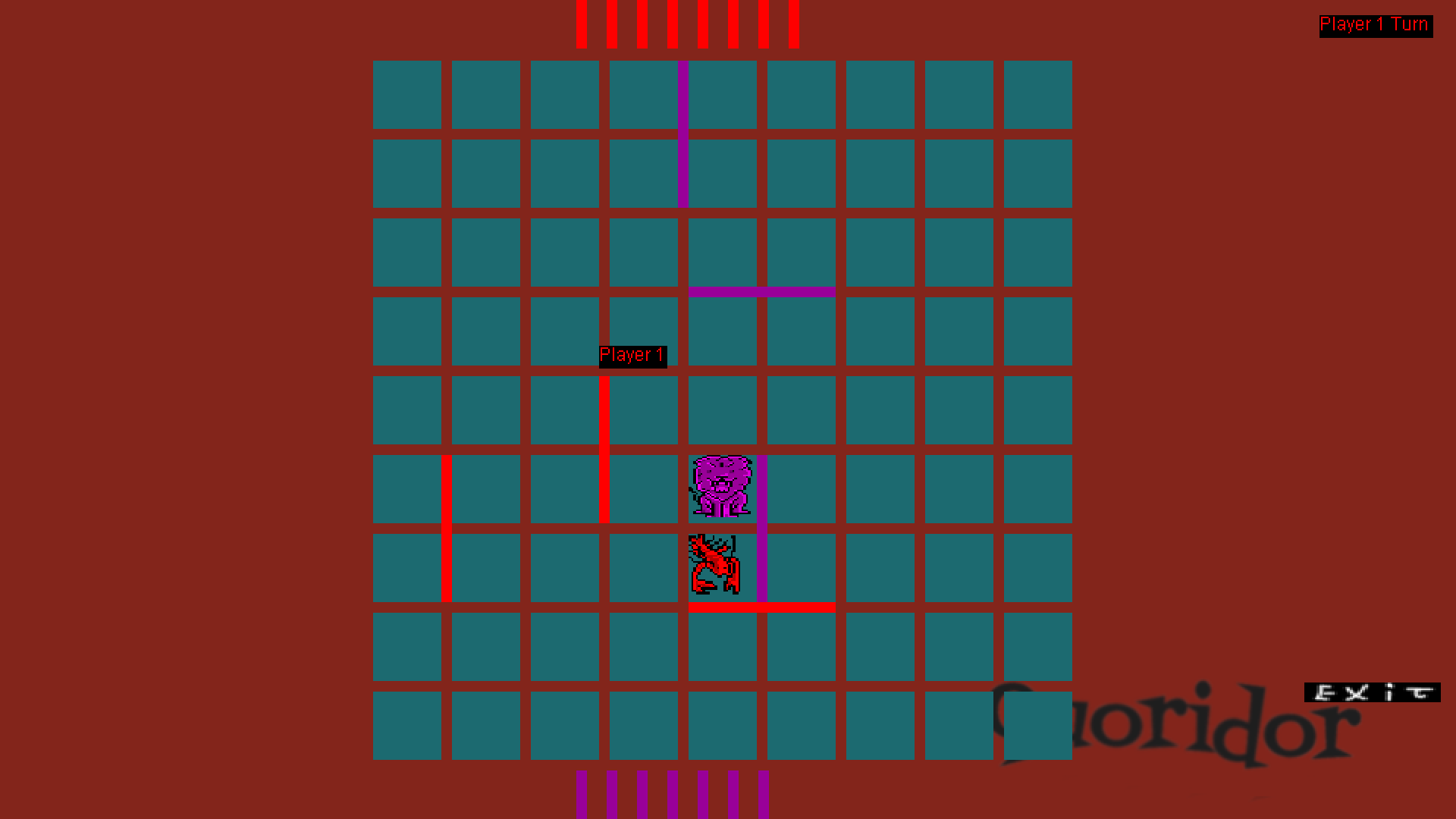


**יכולות:**

1. המערכת מאפשרת לבחור מי יהיה שחקן אנושי ומי שחקן של המחשב.
2. המערכת מאפשרת לבחור את הדמות והצבע של כל שחקן ובנוסף את כמות השחקנים.
3. המערכת מאפשרת משחק בין שחקנים אנושים לשחקנים ממוחשבים.
4. המערכת מאתחלת את פונקציות בדיקת המהלכים, התורות והשחקנים ברגע בחירת השחקנים.
5. המערכת לא מאפשרת לשחקן להתקדם לכיוון שיש בו מחסום.
6. המערכת לא מאפשרת לשחקן לצאת מגובלת הלוח.
7. המערכת מאפשרת לשחקן להתקדם מעל היריבים במידה ואין מאחוריהם מחסום.
8. המערכת מאפשרת לשחקן להתקדם באלכסון אם היריב ניצב מולו ויש מאחוריו מחסום.
9. המערכת מעבירה את התור לשחקן הבא לאחר שהשחקן הנוכחי שיחק והמהלך שלו היה חוקי.
10. המערכת מאפשרת לשחקן למקם מחסומים במקומות חוקיים.
11. המערכת בודקת שהשחקן והיריב מקמו את המחסום במקום חוקי ואם לא מחזירה להם אותו.
12. המערכת מחשבת ומזהה מהו המהלך הכדאי ביותר עבור המחשב, בהתחשב במהלך הטוב ביותר עבור השחקנים האחרים ומצבם ובכך היא מגבירה את הסיכוי לנצח.

**הדגמה של דילוג מעל יריב:**(מצב התחלתי בצד שמאל, צד יד- לאחר הדילוג)   **הדגמה של תזוזה באלכסון במידה ויש מחסום מחאורי היריב:**(מצב התחלתי בצד שמאל, צד ימין- לאחר הדילוג) 

**הדגמה של בחירה היכן למקם מחסום:**

(הUI מסמן היכן המחסום שאתה רוצה למקם נמצא לאחר שממקמים אותו הUI חוזר לשחקן)

**אמצעי ההתקדמות הללו הינם חלק מחוקי המשחק האמיתי ואני ניסיתי לקיים אותם בפרויקט שלי בדרך שתהיה ייחודית לאופן ההתנהלות של המשחק.**A picture containing text, crossword puzzle

Description automatically generated

## **8. מבנה נתונים**

**מבנה הנתונים** שבו השתמשתי לצורך המשחק הינו **מטריצה** בגודל 9X9, כאשר כל תא במטריצה הוא בלוק עם תכונות שמאפיינות את מצבו.  
במחלקה Board התאים במטריצה הם מסוג Block- מהווים את הצד הלוגי של כל תא. (תא מכיל מידע הכיוונים שחסומים ואם יש שחקן באותו תא) ללוח יש גם רשימה של מחסומים שהוצבו בו.

\*התאים בלוח מאותחלים בצורה שהתאים שבקצבות הלוח חסומים בכיוונים שמחוץ ללוח.

במחלקה Graph  
נבנה גרף שמשקף את לוח המשחק לאורך המשחק ומתעדכן לפי מצב המשחק.

במחלקה Players  
יש מערך סטטי מטיפוס Player שהוא מכיל את השחקנים ואת המידע על השחקנים השימוש בערך זה הוא על מנת שתוכל להיות גישה לשחקנים גם ממחלקות אחרות.

לדוגמא על מנת לשנות את הכמות והתכנות כמו האם CPU או להשתמש במידע עליהם בשביל אסטרטגיה של הCPU.

//הבדל עם PLAYER

במחלקה Grahpics ישנם מערכים סטטים שמכילים את התמונות של השחקנים ואת הצבעים שלהם.

במחלקה SettingsState

קיימת מטריצה שהינה מציגה את דמויות המשחק שניתן לבחור.

## **9. תרשים מחלקות**

Diagram, engineering drawing

Description automatically generated

## **10. ארכיטקטורה Top-Down Level Design**

Timeline

Description automatically generated

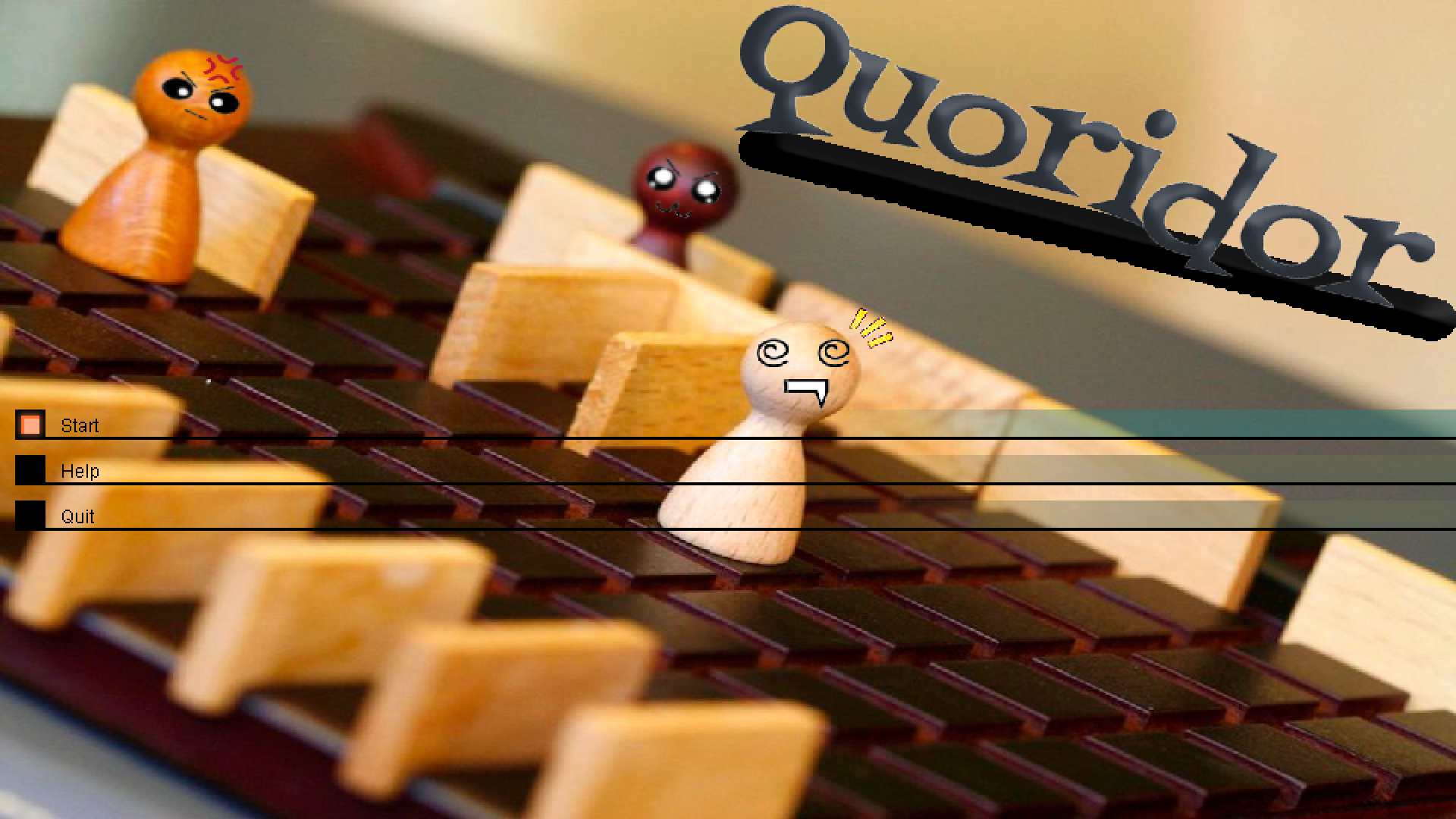
## **11. תיאור סביבת העבודה ושפת התכנות**

יישום זה פותח בסביבת העבודה :  


היישום נכתב בשפת ג'אווה (JAVA)

## **12. תיאור הממשקים**

**תפריט פתיחה:**

 **לחיצה על:**  
**1) Start-** מעביר לדף ההגדרות  
**2) Instructions-** מעביר לדף הוראות  
**3) Exit-** תוביל ליציאה מהמשחק

**דף הגדרות:**

בוחרים שחקן בלחיצה על אנטר.מוסיפים שחקן בלחיצה על הפלוס ומורידים שחקן בלחיצה על המינוס.משנים את הצבע של השחקנים בלחיצה על COLOR

לאחר שבוחרים את כמות השחקנים הרצויה לוחצים על אנטר פעם נוספת על מנת לאשר את הבחירה

ניתן לבטל או לחזור אחורה בלחיצה על Esc



ניתן לשנות שחקן ל"שחקן אנושי" או לא "CPU" בלחיצה על Player/CPU  
**חלון הוראות:**

לחיצה על החצים מעבירה לעמודים הבאים  
  
A picture containing graphical user interface

Description automatically generated

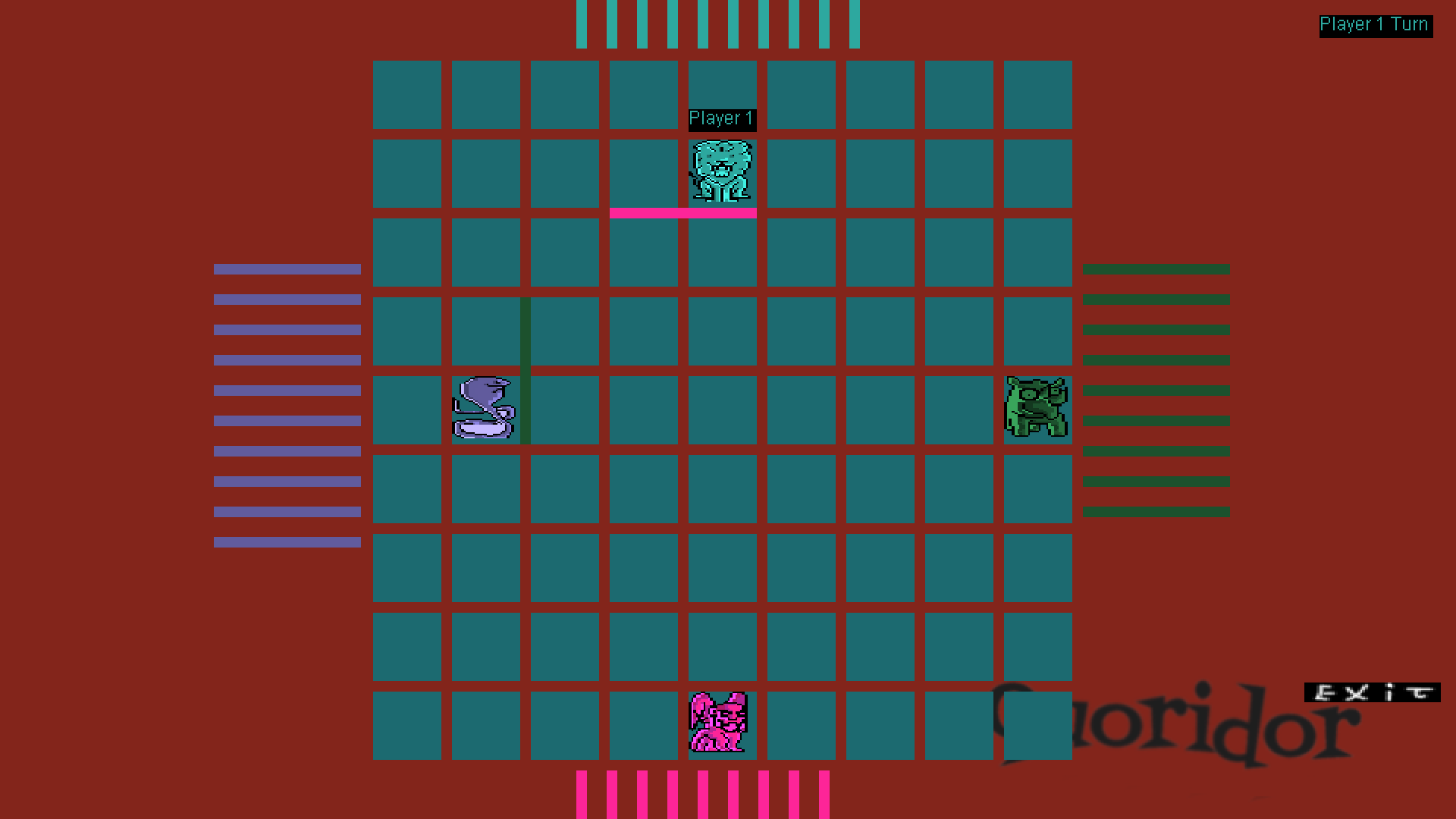
Diagram

Description automatically generated with medium confidence Diagram, engineering drawing

Description automatically generated Diagram, engineering drawing

Description automatically generated

**חלון המשחק:**



## **13. אלגוריתם ראשי**

האלגוריתם לבדיקת תקינות צעד מתחלק בכמה צורות.

בדיקת התקינות של אם שחקן יכול להתקדם ל- נמצאת במחלקת השחקן היא בודקת האם המשבצת בכיוון שעליו הוא רוצה לפנות אינה חסומה בכיוון הנגדי לכיוון שלו, כלאמר אם הוא רוצה להתקדם ימינה האם המשבצת אינה חסומה מצד שמאל.

בדיקת תקינות של מחסומים- ישנם שתי פעולות תקינות מחסומים הראשונה האם המקום בו השחקן רוצה להציב את המחסום אינו חסום כבר על ידי מחסום אחר.

ובדיקה האם מיקום המחסום אינו מונע מאחד השחקנים להגיע ליעד שלו, אלגוריתם זה פועל באמצעות גרף וBFS, האלגוריתם עובר על כל השחקנים ובודק שאכן ישנה דרך בה כל שחקן יכול להגיע ליעד שלו.

סיבוכיות האלגוריתם:

 O(V+E)

V-vertex

E-edge

public class BFS {

//is there away to reach the goal/ did it reach it

private static boolean *\_reachEnd*;

//visited cells matrix

private static boolean *\_visited*[][];

//how many move/ laps the algorithm took

private static int *movCount*;

//how many nodes/cells are currently in progres to reach the end in the current move/step/lap

private static int *nodesLeftInLayer*;

//how many nodes we can check in the next lap/step/mov

private static int *nodesInNextLayer*;

שם הפונקציה: BFS

תיאור הפונקציה:הפונקציה מאתחלת את כל המשתנים במחלקת BFS ומכינה אותם לחיפוש חדש בלוח מהנקודה שהתקבלה לנקודה שהתקבלה בגרף המתקבל

טענת כניסה:נקודה להתחיל ממנה את החיפוש ונקודה שעליה צריך להגיע והגרף שבו מתחולל החיפוש

טענת יציאה: המרחק בין שני הנקודות , במידה ואינו קיים מינוס אחז

רמת סיבוכיות:O(1)

//init all the parameters for a new search, then call the actual BFS algorithm

public static int bFS(int row,int col,Point goal, Graph g) {

*movCount* = 0;

*nodesLeftInLayer* = 1;//current point

*nodesInNextLayer* = 0;

*\_reachEnd* = false;

*\_visited* = new boolean[g.get\_rows()][g.get\_cols()];

//init visited matrix cells to false(we havent visited any cell yet)

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

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

*\_visited*[i][j] = false;

}

}

//call the actual BFS

boolean flag = *solve*(row, col,goal, g);

//if the algorithm found way to reach the end, return how many steps it took

if (flag)

return *movCount*;

//if it didnt find a way then return -1

return -1;

}

שם הפונקציה: solve

תיאור הפונקציה: פונקציה זו הינה מממשת את את האלגוריתם החיפוש של BFS

טענת כניסה: שורה ועמודה להתחיל ממנה, נקודה שעליה צריך להגיע ואת הגרף שבוא מתחולל החיפוש

טענת יציאה: האם הפוקציה הצליחה להגיע לנקודה המבוקשת

רמת סיבוכיות: O(V+E)

private static boolean solve(int row, int col,Point goal, Graph g) {

// remove 0 and add last = Queue

ArrayList<Integer> rq = new ArrayList<Integer>();//rows Queue

ArrayList<Integer> cq = new ArrayList<Integer>();//cols Queue

//add current point

rq.add(row);

cq.add(col);

//mark this point as visited

*\_visited*[row][col] = true;

//while there arent any points left and we havent reach the goal

while (!*\_reachEnd*&& rq.size() > 0 && cq.size() > 0) {

//check the node in the current layer

row = rq.remove(0);

col = cq.remove(0);

//if it acctualy the goal than exit with true

if (row == goal.x||col==goal.y) {

*\_reachEnd*=true;

return true;

}

//check what node's neighbours we can add to the next layer

*explore\_neighbours*(row, col, rq, cq, g);

//moving to the next node in this layer

*nodesLeftInLayer*--;

//if the layer ended we need to check their childern

if (*nodesLeftInLayer* == 0) {

//the next layer turning into the current layer

*nodesLeftInLayer* = *nodesInNextLayer*;

//init next layer counter

*nodesInNextLayer* = 0;

//increase progres move

*movCount*++;

}

}

//if the algorithm reach this point it means it didnt find a way to reach the goal

return *\_reachEnd*;

}

שם הפונקציה: explore\_neighbours

תיאור הפונקציה: הפעולה בודקת את כל השכנים של הנקודה שהתקבלה ומציבה ברשימת השורות והעמודות האם קיימים שכנים שלא ביקרנו בהם

טענת כניסה: שורה ועמודה ליבדוק עליה את השכנים , רשימת עמודות ושורות, והגרף

טענת יציאה: VOID

רמת סיבוכיות:O(V) כאשר V זה כמות השכנים

//a function that adds the nodes the algorithm can visit in the next layer

private static void explore\_neighbours(int row, int col, ArrayList<Integer> rq, ArrayList<Integer> cq, Graph g) {

ArrayList<Edge> e = g.getGraph()[row][col].getAdjacencies();//array of edges/cells that the player can go to from the row col- cell

for (int i = 0; i < e.size(); i++) {

//get the position of the neighbour

int rr = e.get(i).getTarget().getI();

int cc = e.get(i).getTarget().getJ();

//if we visited the node then we shouldnt add it to the next layer again (we already treated this node)

if (*\_visited*[rr][cc])

continue;

//if we didnt visited the node then add its point

rq.add(rr);

cq.add(cc);

//mark as visited

*\_visited*[rr][cc] = true;

//the nodes in the next layer have been increased

*nodesInNextLayer*++;

}

}

}

## **14. תיאור המחלקות הראשיות**

1. **מחלקת הפאנאל GamePanel**

- יורש מJPanel, מממש את הממשק KeyListener, Runnable ואת MouseListener דרך MouseAdapter  
- טיפול בTHREAT ובמנהל הדפים- עידכון של הדף הנוכחי וה"ציור" שלו על הפאנאל

1. **מחלקת** GameState-מחלקת עמוד

**-מחקלה אבסטרקטית שמכילה פעולות אבסטרקטיות ופראמטרים שהם מייצגים את התכונות של "דף" /"עמוד" במשחק**

**-כל עמוד במשחק יורש מהמחלקה הזו**

1. מחלקת מנהל הדפים GameStateManager

-מחלקה שאחרית למעבר בין העמודים של המשחק ודואגת לטעון ולעלות את כל הדפים והמעברים בייניהם

1. מחלקת הכפתורים Keys

-מחלקה שמנהלת את לחיצות הכפתורים במשחק על מנת ליצור גנריות לכפתורים המיועדים למשחק ולבדיקות של איזה כפתור נלחץ

1. מחלקת כלי לוח BoardItem

-המחלקה מחילה נתונים בסיסים של כלי שנימצא על הלוח כמו שורה עמודה ואת המיקום הגרפי של האובייקט X,Y

1. מחלקת לוח המשחק, יורשת ממחלקת GameBoard

-הינה אחראית על מהלך המשחק ועל התנהלותו דואגת לחיבור של כל המחלקות האחרות ולמימושם כדי שהמשחק יתקיים

1. מחלקת שחקנים Players

-מחלקה סטאטית המחילה משתנים ופעולות סטטיות על הגדרות השחקנים, המחלקה מכילה מערך של השחקנים שמשתתפים במשחק. השימוש במחלקה הוא לצורך טיווח בין מחלקות, השימוש במחלקה נותן את האפשרות להגדיר את הגדרות השחקנים כמו האם השחקן אנושי אוCPU או את כמות השחקנים בעמוד ההגדרות לפני יצירת הGamePanel. וגם מאפשר להפנות לכל השחקנים ממחלקות אחרות לצורכים אחרים כמו בדיקות לוגיות של מהלכים או תהליך הבינה מלאכותית.

1. מחלקת תורות Turns

-מחלקה סטאטית שמנהלת את התקדמות התורות מכילה משתנים כמו את השחקן שמשחק כעט את מספר השחקן,כמות השחקנים , את המהלך שעשה

0- עוד לא שיחק

1-התקדם

2-שם מחסום

1. המחלקה גרף Graph

-מחלקה של גרף המייצג את לוח המשחק בגרף. הגרף קיים על מנת לגשת ללוח המשחק בנקודת מבט נוחה יותר . עם הגרף אנחנו נוכל לעשות בדיקות וחיפושים בפשטות בלי ליפגוע בלוח המשחק

-הגרף נועד כדי לעזור לנהל את הלוגיקה של המשחק

1. המחלקה AI

-המחלקה שמנהלת את הבינה מלאכותית, במחלקה זו ניגש בכל תור של CPU כאשר נרצה להחליט על המהלך של היריב.

1. המחלקה Grahpics

-המחלקה אחראית לכל הנראות הגרפית של המשחק על הפאנאל

1. המחלקה SettingsState יורשת מ GameState

-המחלקה מנהלת את בחירת ההגדרות של השחקן, את כמות השחקנים, איזה שחקן הינו אנשוי ואיזה שחקן הינו CPU, בחירת הדמות של השחקן והצבע שלו.

1. מחלקת הלוח Board

-מחילה מטריצה של תאים בלוח ורשימה של מחסומים הממוקמים בלוח

-המחלקה מאתחלת את המיקום של שחקן ואת המחסום שלו במידה וינסה לשים

-המחלקה גם דואגת לעדכן את התאים שהדרך בהם חסומה ואחראית לבדוק האם שחקן יכול להתקדם למקום מסויים בלוח

1. המחלקה שחקן Player יורשת מכלי לוח

-מחלקה לוגית שמייצגת את השחקן המחלקה אחראית לפעולות שהוא יכול לבצע כמו להתקדם או לשים מחסום

1. מחלקת מחסום Barrier יורשת מכלי לוח

-המחלקה מייצגת את הלוגיקה של מחסום מייצג באיזה צורה המחסום חוסם אופקי או אנכי ואת המיקום שלו

1. המחלקה כיוונים Directions

-מחלקה אשר ממשת ENUM של הכיוונים במשחק ופניה לכיוון מסויים עם שני מטריצות של התקדמות בX והתקדמות בY

-כל אינדקס מקביל במערכים מייצג כיוון התקדמות על הלוח(לזוז למעלה מתקדמים -1 בY ו0 בX)

להלן פירוט של כל אחת מהמחלקות:

**1.** מחלקת לוח משחק- GameBoard  
public class GamePanel extends JPanel implements Runnable, KeyListener

**מאפיינים:**

// dimensions

public static int *WIDTH* = 1;

public static int *HEIGHT* = 1;

public static final int ***SCALE*** = 2;//make images bigger

// game thread

private Thread \_thread;

private boolean \_running;

private int \_FPS = 60;

private long \_targetTime = 1000 / \_FPS;

// image

private BufferedImage \_image;

private Graphics2D \_g;

// game state manager

private GameStateManager \_gsm; private GameStateManager \_gsm;

**שם הפונקציה:** public GamePanel(int w, int h)  
תיאור הפונקציה: בנאי המאתחל את גודל המסך לגודל שקיבלה המחלקה (הגודל היינו בגודל של רזולוציאת המסך),  
ומסדר את הפוקוס לפריים על מנת שהפריים יוכל לקלוט לחיצת כפתורים.

**פונקציות:**

טענת כניסה: אורך ורוחב המסך

טענת יציאה:VOID

**סיבוכיות:** O(1)   
 **שם הפונקציה:** public void addNotify()  
תיאור הפונקציה: הופך את הרכיב לתצוגה על ידי חיבורו למשאב מסך מקורי. שיטה זו נקראת באופן פנימי על ידי הtoolkit.

addNotify () משמש רכיבים ברמה נמוכה כדי לקיים אינטראקציה עם העמית הבסיסי במערכת ההפעלה כך שמשהו באמת יקרה, ולא רק תמונות על המסך.

בנוסף, מגדיר את משדרי האירועים KeyListener וAdapter של MouseListener וגם מאתחל את החוט/THREAT של המשחק.

**סיבוכיות:** O(1)   
 **שם הפונקציה:** public void run()  
תיאור הפונקציה: פעולה השייכת לממשק Runnableומופעלת בכל רגע נתון שהתוכנית עובדת.  
מתחילה בלאפס את הפאנאל עם הפעולה init .

המטרה העיקרית של הפונקציה הוא: לעדכן את כל נתוני המשחק והעמוד שפתוח, לצייר את כל הנתונים שמתרחשים בעמוד במשחק ולבסוף לצייר את השיכלול למסך.

לאחר מכן הTHREAD "נח" לכמות זמן שנקבע וחושבה על ידי משתנים גלובאלים(המשחק שלי מתעדכן 60 FPS פריימים בשניה)

**סיבוכיות:** הסיבוכיות הינה תלוי בסיבוכיות שקוראת בעמוד הנוכחי ומתקיימת כל עד הפריים פתוח  
 **שם הפונקציה:** private void update()  
תיאור הפונקציה: מעדכנת את הנתונים במנהל המשימות.

**סיבוכיות:**תלויה בעמוד הנוכחי .

**שם הפונקציה:** private void draw()  
תיאור הפונקציה: מעבירה את כל הנתונים הגרפים מהעמוד הנוכחי למשתנה הגרפי של הפאנאל.

**סיבוכיות:**תלויה בעמוד הנוכחי.

**שם הפונקציה:** private void drawToScreen()  
תיאור הפונקציה: מצייר את העמוד הנוכחי בלוח הפאנאל.

השימוש במשתנה גרפי נועד בשביל שליטה יותר גדולה בחלק בגרפי.

במקרה שלי השתמשתי במשתנה גרפי שהעברתי עליו את כל נתוני העמוד הנוכחי ולאחר מכן עשיתי בו שימוש בצורה שרציתי (להגדיל את כל הפרטים בציור)

**סיבוכיות:**O(1).

**שם הפונקציה:** public void keyPressed(KeyEvent key)ו

public void keyReleased(KeyEvent key)  
תיאור הפונקציה: מעדכנות במחלקת המקשים האם נלחץ כפתור מסויים או ששוחרר.

טענת כניסה: משתנה המכיל את המידע על מה שקרה באירוע בלחיצה

טענת יציאה: VOID

**סיבוכיות:**O(1).

**2.** מחלקת אובסטרקטית מצב/עמוד- GameState  
public abstract class GameState

**מאפיינים:**

protected GameStateManager \_gsm;

מנהל המשימות

**פונקציות אובסטרקטיות:**

public abstract void init()

מתרת הפעולה: לאפס את נתוני העמוד

public abstract void update()

מתרת הפעולה: לעדכן את נתוני העמוד

public abstract void draw(Graphics2D g)

מתרת הפעולה: לצייר את הנתונים הגרפים של העמוד

public abstract void handleInput()

מתרת הפעולה: לסדר ולדאוג ללחיצת המקשים במחלקת KEYS לעדכן את הנתונים בקשר ללחיצת כפתור או עזיבה

public abstract void MouseClicked(MouseEvent e)

מתרת הפעולה: לעדכן את נתוני לחצית הכפתור והפעלה בהתאם (לחיצה על כפתורים במשחק בדכ)

public abstract void MouseScrolled(MouseWheelEvent e)

מתרת הפעולה: פעולה לא עובדת..

**3.** מחלקת מנהל העמודים/מצבים- GameStateManager

public class GameStateManager

**מאפיינים:**

//game states array

private GameState[] \_gameStates;

//the index of the current state/page

private int \_currentState;

//load screen image

public Background \_loadScreen;

//enum of game states

public static int *\_statCounter* = 0;

public static final int ***MENUSTATE*** = *\_statCounter*++;

public static final int ***SETTINGS*** = *\_statCounter*++;

public static final int ***GAMESTATE*** = *\_statCounter*++;

public static final int ***NUMGAMESTATES*** = *\_statCounter*;

**שם הפונקציה:** public GameStateManager()  
תיאור הפונקציה: בנאי המאתחל את מערך עמודי המשחק ומאתחלת את תמונת הטעינה (לרוב לא יהיה שימוש בה כי טעינת העמודים אינה דורשת מהמערכת יותר מידי אך היא קיימת כדי לשמור על אסתטיקה במשחק).

**פונקציות:**

הפעולה גם מציבה במצב/עמוד הנוכחי את הדף התפריט-MENU מכיוון שזה הוא העמוד הראשון שהמשתמש מגיע עליו כשהוא מפעיל את המשחק.

**סיבוכיות:** O(1)

**שם הפונקציה:** public void setState(int state)  
תיאור הפונקציה: המטודה מקבלת מצב/עמוד (לפי הENUM של המצבים הקיימים) ומבטלת/מוחקת את העמוד הנוכחי ולאחר מכן משנה את אינדקס העמוד הנוכחי לעמוד שהתקבל וטוענת אותו.

טענת כניסה: מספר מצב/דף

**סיבוכיות:** O(1)

**שם הפונקציה:** private void loadState(int state)  
תיאור הפונקציה: המטודה יוצרת/ טוענת את העמוד באינדקס שקיבלה.

טענת כניסה: מספר מצב/דף

**סיבוכיות:** תלוי בעמוד שנטען

**שם הפונקציה:** private void unloadState(int state)  
תיאור הפונקציה: לפרוק/למחוק את העמוד באינדקס שהתקבל (הפונק' רק משווה את המצביע עליו לNULL מכיוון שהסביבת עבודה הינה בגאבה, וקיים grabge collector ואין על המתכנת לטפל בזיכרון התכונית).

טענת כניסה: מספר מצב/דף

**סיבוכיות:** O(1)

**שם הפונקציה:** public void update()  
תיאור הפונקציה: מעדכנת את נתוני העמוד.

**סיבוכיות:** תלוי בעמוד הנוכחי

**שם הפונקציה:** public void draw(Graphics2D g)  
תיאור הפונקציה: מעדכנת את הנתונים הגרפים של העמוד הנוכחי.

טענת כניסה: משתנה גרפי

**סיבוכיות:** תלוי בעמוד הנוכחי

**4.** מחלקת מקש- Keys  
public class Keys

**מאפיינים:**

//my game buttons enum

private static int *\_counter* = 0;

public static int *UP* = *\_counter*++;

public static int *LEFT* = *\_counter*++;

public static int *DOWN* = *\_counter*++;

public static int *RIGHT* = *\_counter*++;

public static int *RB* = *\_counter*++;

public static int *BB* = *\_counter*++;

public static int *ENTER* = *\_counter*++;

public static int *ESCAPE* = *\_counter*++;

//the number of keys in my game

public static final int ***NUM\_KEYS*** = *\_counter*;

public static boolean *keyState*[] = new boolean[***NUM\_KEYS***];

public static boolean *prevKeyState*[] = new boolean[***NUM\_KEYS***];

**שם הפונקציה:** public static void keySet(int i, boolean b)  
תיאור הפונקציה: פעולה המציבה בתא של כפתור בHASH המתקבל(i) ומשתנה בוליאני שאומר האם הכפתור לחוץ או שנעזב- הפעולה ניקראת בkeyPressed וב keyReleased ומציבה אמת כאשר כפתור נלחץ ושקר כאשר כפתור נעזב.

**פונקציות:**

טענת כניסה: משתנה i אשר ערכו הוא ערך HASH של תו במקלדת ומשתנה בוליאני שאומר האם הכפתור לחוץ או שנעזב

טענת יציאה: VOID

**סיבוכיות:** O(N)כאשר N מייצג את כמות הכפתורים במשחק שלי

**שם הפונקציה:** public static boolean isPressed(int i)

תיאור הפונקציה: הפעולה בודקת האם הכפתור שהתקבל (לפי הENUM של הכפתורים) לחוץ או נלחץ.

טענת כניסה: משתנה i אשר ערכו הוא ערך HASH של תו במקלדת

טענת יציאה: האם הכפתור נלך (כלאמר נלחץ בפריים הנוכחי ולא לחוץ)

**סיבוכיות:** O(1)

**שם הפונקציה:** public static void update()

תיאור הפונקציה: הפעולה מעדכנת את הכפתורים הPREV KEY כלאמר לאחר שנלחץ כפתור לראשונה הפונקציה מעדכנת את המשתנה שאומר שהכפתור כבר נלחץ(לאחר שמשחררים אותו תעדכן שהוא נעזב).

**סיבוכיות:** O(N)כאשר N מייצג את כמות הכפתורים במשחק שלי

**5.** מחלקת כלי לוח (אובטרקטית)- BoardItem  
public abstract class BoardItem

**מאפיינים:**

protected Board \_board;

protected int \_col;

protected int \_row;

protected int \_x;

protected int \_y;

**פונקציות SET:** public void setPosition(int x, int y)

**פונקציות:**

public void setRowCol(int row, int col)

**שם הפונקציה:** public abstract void update()  
תיאור הפונקציה: פונקציה אובסטרקטית שמעדכנת את כלי הלוח.

טענת כניסה: שורה ועמודה , איקס וואי. להציב במשתנים של המחלקה

טענת יציאה: VOID

**סיבוכיות:** תלוי בירוש

**6.** מחלקת עמוד/מצב לוח המשחק- GameBoard  
public class GameBoard extends GameState

**מאפיינים:**

private Background \_bg;

private Board \_board;

private Graph \_graph;

private int \_playerWon = 0;// 0 -havent win yet

private Img \_exitButton;

private boolean pause = false;

private PopMessage \_restart;

private PopMessage \_changeSettings;

private PopMessage \_playerWonMessage;

private PopMessage \_playerWonNumerMessage;// init to 1

private PopMessage \_wonMessage;

**פונקציות:**

**שם הפונקציה:** public GameBoard(GameStateManager gsm)  
תיאור הפונקציה: בנאי המאתחל מנהל המצבים/דפים של העמוד הנוכחי וקורא לinit.

**סיבוכיות:** החלק שקורה בפעולה עצמה O(1) . **שם הפונקציה:** public void init()  
תיאור הפונקציה: הפוקציה מאפסת את כל נתוני המשחק להתחלת משחק חדש.

**סיבוכיות:** החלק שקורה בפעולה עצמה O(N) כאשר N הוא מס השחקנים אך יצירת הלוח והגרף או יצירת/ איפס של אובייקטים אחרים הם סיבוכיות בפני עצמן **שם הפונקציה:** public void setPlayersPositions()  
תיאור הפונקציה: פונקציה מאפסת את המקום של השחקנים ואת היעד שהם רוצים להגיע עליו.

**סיבוכיות:** O(N) כאשר N הוא מס השחקנים **שם הפונקציה:** public void update()  
תיאור הפונקציה: זהוי פונקציה אשר העמוד ירש מGameState הפוקנציה דואגת לעדכן את התנהלות המשחק כמו התזוזה של השחקנים קבלת INPUT מהמשתמש המהלך של המשתמש או עידכון המהלכים של AI

ולבדוק האם אחד השחקנים נצח במשחק

**סיבוכיות:** O(N)כאשר N הוא מס השחקנים, הסיבוכיות הינה תקפה רק למתרחש בפונקציה עצמה, הסיבכויות של שאר הפונקציות שניקראות מזהו פרטו או יפרטו בהמשך **שם הפונקציה:** private void Won(int i)  
תיאור הפונקציה: הפונקציה מציגה הודעת נצחון על המסך ומעדכנת את ההודעות המוצגות על המסך.

טענת כניסה: אינדקס של שחקן מנצח

טענת יציאה: VOID

**סיבוכיות:** O(1)

**שם הפונקציה:** public void draw(Graphics2D g)  
תיאור הפונקציה: פונקציה מציירת לפאנל את התרחשות המשחק, השחקנים, המחסומים, הלוח וUI של השחקן הנוכחי.

טענת כניסה: משתנה גראפי

סיבוכיות: O(1)

**שם הפונקציה:** public boolean nextTurn()  
תיאור הפונקציה: הפעולה מעבירה את התור לשחקן הבא במידה והפעולה האחרונה שהשחקן הנוכחי עשה היית תקינה.

טענת יציאה: האם התור יכול לעבור לשחקן הבא

**סיבוכיות:** O(1)  **שם הפונקציה:** public boolean handleLogic()  
תיאור הפונקציה: פונקציה הפעולה בודקת האם הפעולה האחרונה שהשחקן הנוכחי ביצע היית תקינה, במידה ולא הפעולה מחזירה לשחקן את המחסום שהוא שם , מאפסת את האזור של המחסום שהסירה מהלוח גם בגרף וגם בלוח עצמו, מאפסת את התור של השחקן ואת הAI ומידעת את מחלקת הAI שהAI כבר ניסה לשים מחסום אך ללא הצלחה

טענת יציאה: האם הפעולה של השחקן האחרון היית חוקית

**סיבוכיות:** O(1) **שם הפונקציה:** public void handleInput() ו

public void MouseClicked(MouseEvent e)  
תיאור הפונקציה: פעולות אשר המחלקה יורשת אותן, בודקות את קלט המשתמש.

**סיבוכיות: בבדיקת המקשים** O(N) כאשר N הוא מס המשקים במשחק ובלחיצת הכפתור O(1)

**7.** מחלקת שחקני המשחק- Players  
public class Players

**מאפיינים:**

// array of the players

public static Player *\_players*[];

**שם הפונקציה:** public static void init(int num)  
תיאור הפונקציה: הפונציה הינה כמו בנאי למחלקה (מכיוון שהינה מחלקה סטטית) מקבלת את מס' השחקנים הרצוי ויוצרת אותם.

**פונקציות:**

טענת כניסה: מאפס את מספר השחקנים במשחק

**סיבוכיות:** O(N)אורך הקלט **שם הפונקציה:** public static void initPlayers()   
תיאור הפונקציה: מטודה המופעלת לאחר לחיצה על משחק חדש, המטודה מאפסת את נתוני השחקנים למשחק חדש.

**סיבוכיות:** O(N) אורך השחקנים

**שם הפונקציה:**

public static void setAI(int num, boolean flag)   
תיאור הפונקציה: הפעולה משנה את השחקן במקום הNUM לAI או לאנושי תלוי בFLAG.

טענת כניסה: מספר שחקן לפי האינדקס ומשתנה אשר אומר אם הוא AI או לא

**סיבוכיות:** O(1)

**שם הפונקציה:**

public static boolean addOrRemovePlayer(int amount)  
תיאור הפונקציה: הפעולה מקבלת את כמות השחקנים החדשה במידה והכמות קטנה יותר הפונציה מסירה את השחקן האחרון והמערך קטן, במידה והכמות גדולה יותר מגדילה את המערך ומעתיקה את השחקנים המקוריים.

**סיבוכיות:** O(N) כאשר N הוא כמות השחקנים החדשה.

טענת כניסה: מספר השחקנים החדש

טענת יציאה: האם מס השחקנים הוגדל

**8.** מחלקת תורות- Turns  
public class Turns

**מאפיינים:**

public static int *\_currentTurn*=-1;//-1 means that the game is done

public static int *numTurns*;//num of turns

private static Player *\_currentPlayer*;//the current player

public static int *\_curAction*=0;//0- none, 1-moved, 2-barrier

**שם הפונקציה:**

**פונקציות:**

public static int getNextTurn() ו

public static int getPrevTurn()  
תיאור הפונקציה: הפונקציה מחזירה את מס האינדקס של השחקן הבא או השחקן הקודם.

טענת יציאה: אינדס השחקן הבא או הקודם בהתאם

**סיבוכיות:** O(1) **שם הפונקציה:** public static void nextTurn()   
תיאור הפונקציה: הפונקציה מאפסת את הנתונים לתחילת תור חדש ומגדירה את השחקן הנוכחי החדש ומאתחלת את התור שלו וגם את הזמן של הAI לחשוב(במידה והשחקן AI).

**סיבוכיות:** O(1)

**9.** מחלקת גרף- Graph  
public class Graph

**מאפיינים:**

private Vertex[][] \_matVert;// the Dijkstra map

private int \_rows;// the rows

private int \_cols;// the colums

public static Board *\_Board*;  
**שם הפונקציה:** public Graph(Board b)  
תיאור הפונקציה: קונסטרקטור, מקבל לוח ,מאפס את גודל הנתונים לפי הלוח ,ויוצר את הלוח המשחק בתור גרף.

**פונקציות:**

טענת כניסה: לוח המשחק

**סיבוכיות:** O(V + E) **שם הפונקציה:** public void initBarrier(Barrier b)   
תיאור הפונקציה: הפונקציה מאפסת את השכנים בגרף באזור המחסום המתקבל (משתמשים בזה לאחר שמציבים מחסום בלוח או מסירים)

טענת כניסה: מחסום

**סיבוכיות:** O(V) כאשר V הוא מס השכנים של התאים.

**10.** מחלקת הבינה מלאכותית- AI  
public class AI

**מאפיינים:**

// length and counter of how long it seems that the player is thinking

private static int *\_AIThink*;

private static Long *\_AIThinkTime*;

private static boolean *\_finishedTurn*;// did the AI finished their act

private static boolean *\_tryedToPutBarrier*;// did the AI tryed to put barrier but the position was ileagal

private static int *\_lastBestRow*[] = new int[5], *\_lastBestCol*[] = new int[5];// last best row and col that were found

// in the algorithm

private static Board *\_board*;// the game board  
**שם הפונקציה:** private static void decideNextAction()  
תיאור הפונקציה: הפונקציה אחראית לשלב את כל הפעולות הבינה מלאכותית, מסדרת את המרחקים הכי קצרים של כל השחקנים במערך מוצאת למי יש את המינימום מרחק לסוף (במידה שלשני שחקנים יש אורך מינימאלי תקח רנדומאלית את אחד מהם), ולבסוף אם השחקן לא יציב מחסום תקדם את השחקן לנתיב הקצר ביותר עבורו.

**פונקציות:**

**סיבוכיות:** O(N)כאשר N הוא מס השחקנים  **שם הפונקציה:**

private static boolean blockShortestPathPlayer(int min, int curturn, int arr[])  
תיאור הפונקציה: במידה והפעולה מוצאת לנכון חוסמת את הדרך הכי קצרה של השחקן עם הדרך הכי קצרה או השחקן השני בעל הדרך הכי קצרה (שהם לא הוא) כאשר מקום החחסימה הינה המקום שהכי יקשה על השחקן להתקדם בו.

טענת כניסה: שחקן בעל כמות הצעדים המינימאלית ,השחקן הנוכחי ומערך אשר מציין את הדרך למטרה של כל אחד מהשחקנים לפי אינדקס

טענת יציאה: האם השחקן שם מחסום או לא

**סיבוכיות:** O(N) אורך השחקנים

**שם הפונקציה:**

private static boolean bestSpotToPutBarrier(int min, int curturn, int[] arr)  
תיאור הפונקציה: הפעולה מוצאת את המקום שיחסום את השחקן MIN הכין שהדרך שלו תיהיה הכי ארוכה משאר המקומות בהתחשבות בכך שהמקום לא ישפיע משמעותית על השחקן ה CURTURN (מקום החסימה הינה חוקי).

טענת כניסה: שחקן בעל כמות הצעדים המינימאלית ,השחקן הנוכחי ומערך אשר מציין את הדרך למטרה של כל אחד מהשחקנים לפי אינדקס

טענת יציאה: האם השחקן שם מחסום או לא

**סיבוכיות:** O(N) כאשר N הוא מס הבלוקים בלוח (הסיבוכיות הינה תקפה למה שקורה בפונקציה עצמה , בהמלך הפוקציה ניקראות פעולות כמו BFS ובדיקה לתקינות הנחת מחסום מה שמוביל לסיבוכיות גבוהה יותר)

**שם הפונקציה:**

private static int shouldPutBarrier(int playerNum, int[] arr)

תיאור הפונקציה: הפעולה מקבלת שחקן ומערך המחרקים של השחקנים, הפעולה מחשבת מתמטית את מצב השחקן לפי שאר התנאים בלוח ומחזירה את המספר שיצא.

טענת כניסה: שחקן ומערך אשר מציין את הדרך למטרה של כל אחד מהשחקנים לפי אינדקס

טענת יציאה: המצב/סטטיסטיקת השחקן

**סיבוכיות:** O(N) כאשר N הוא מס השחקנים.

**שם הפונקציה:**

private static int getPlayerShortestPathLen(Player p, int num)

תיאור הפונקציה: הפעולה מקבלת שחקן ואינדקס, הפעולה מוצאת את המרחק הכי קצר שיש לשחקן על מנת להגיע לסוף, הפעולה בודקת מבין כל התאים שיש לו נגישות עליהם איזה מהם הוא התא שהיא כדאי לו להתקדם עליו, שומרת את ה ROW ו COL של התא ומחזירה את האורך ליעד של השחקן מהתא הזה.

טענת כניסה: שחקן ואידנקס להציב בו את תוצאות הBFS במערכים של המחלקה

טענת יציאה: המרחק הקצר ביותר של השחקן לסוף

**סיבוכיות:** O(N) כאשר N הוא מס השחקנים.

**11.** מחלקה גרפית - Graphics  
public class Graphics

**מאפיינים:**

// image of a block

public static final Img ***\_block*** = new Img("/Board/block.png", 1, 1, Block.***size***, Block.***size***);

// players sprites

public static BufferedImage *player*[];

// the size of player

private static int *\_playSize* = Block.***size***;

public static String *numbers*[];// path to numbers imges

// players colors

public static Color *\_playerColor*[];

**שם הפונקציה**public static void setPlayers()

**פונקציות:**

תיאור הפונקציה: הפונציה מעדכנת את המערכים של המחלקה לפי גודל וכמות השחקנים שמשחקים במשחק.

**סיבוכיות:** O(N)כאשר N הוא מס השחקנים  **שם הפונקציה:**

public static void drawCurrPlayerUI(Graphics2D g)  
תיאור הפונקציה: הפונציה מציירת מעל השחקן הנוכחי את המספר שלו.

טענת כניסה: משתנה גראפי

**סיבוכיות:** O(1)

**שם הפונקציה:**

public static void drawCurrPlayerTurn(Graphics2D g)

public static void drawPlayer(int x, int y, int i, Graphics2D g)  
תיאור הפונקציה: הפעולות מציירות את השחקן ואת מס השחקן הנוכחי

טענת כניסה: משתנה גראפי, בשני: איקס ווואי של השחקן בפאנאל וI שמציין את מס השחקן

**שם הפונקציה:**

public static void drawPlayerBarriers(Graphics2D g, int count, Point p, int dx, int dy, int width, int height,int playerNum)

תיאור הפונקציה: הפעולה מציירת את תחנת המחסומים של שחקן מהנקודה P לכיוון של DX וDY עם מרווח בניהם לפי מה שהתקבל בצבע של השחקן הנוכחי ובכמות מחסומים שיש לו.

טענת כניסה: משתנה גראפי כמות מחסומים שנישארו, נקודת התחלה של תכנת המחסומים, כיוון שעליו מתקדמים במחסומים, הרווחים ביניהם של שורות ועמודות ומס השחקן אשר המחסומים שלו

**סיבוכיות:** O(N) כאשר N הוא מס המחסומים של השחקן.

**שם הפונקציה:**

public static void drawBlock(Graphics2D g, int x, int y)

תיאור הפונקציה: מצייר בלוק במקום נתון.

טענת כניסה: משתנה גראפי איקס וואי על הפאנאל

**סיבוכיות:** O(1).

**שם הפונקציה:**

public static void drawBoard(Graphics2D g,

int col, int row)

תיאור הפונקציה: מצייר את הלוח.

טענת כניסה: משתנה גראפי שורה ועמודה

**סיבוכיות:** O(N) כאשר N הוא מספר הבלוקים בלוח.

**שם הפונקציה:**

public static void drawBarriers(ArrayList<Barrier> obstacles, Graphics2D g)

תיאור הפונקציה: מצייר את המחסומים מהרשימה שהתקבלה.

טענת כניסה: משתנה גראפי רשימה של מחסומים

**סיבוכיות:** O(N) כאשר N הוא מס המחסומים.

**שם הפונקציה:**

public static void drawBarierr(Graphics2D g, int x, int y, int width, int height, int num)

תיאור הפונקציה: מצייר מחסום במקום נתון.

טענת כניסה: משתנה גראפי איקס וואי על הפאנאל , אורך רוחבת ומס שחקן

**סיבוכיות:** O(1).

**12.** מצב/דף ההגדרות- SettingsState  
public class SettingsState extends GameState

**מאפיינים:**

// images of all the optional players\sprites to play as

private Img \_sprites[][];

private ColorPicker \_cp;// tool for choosing a color

// imgs size

private final int imgBoxWidth = 30;

private final int imgBoxHeight = 51;

// current row and col of player

private int \_currentRow = 0;

private int \_currentCol = 0;

// current players in the game

private int \_numOfPlayers = 2;

// the player who we set their settings for

private int \_currPlayerSettings = 1;

private boolean canStart = false;// if we set the settings for all the player we can start the game

// images and graphics

private Font \_font;

private Font \_bigFont;

private Img \_exitButton;

private Img \_AddRemovePlayer[];

private Img \_AI[];

private Img \_red;

private Img \_black;

private Img \_changeColor;

private Background \_bg;

**פונקציות:**

public void draw(Graphics2D g)

מצייר את הנתונים של הדף

public void init()

מאפס את הנתונים של הדף

**שם הפונקציה**private void select(boolean exit)

תיאור הפונקציה: הפונקציה מקבלת משתנה בולאיני שאומר האם הבחירה שנעשת נעודה לביטול או לאישור

במידה והיא לביטול אם קיים יותר משחקן אחד שההגדרות שלו כבר אשרו אז מבטלת את ההגדרות של השחקן האחרון שהגדרות שלו נבחרו, אם לא אושר עוד שחקן אחד הפונקציה תחזיר את המשתמש לעמוד הראשי.

במידה והבחירה שנעשת היא לאישור, אם כל ההגדרות של כל השחקנים המשתתפים במשחק אושרו אז הפונקציה תעביר את המשתמש לדף/מצב המשחק, אם טרם כל ההגדרות של השחקנים אשרו אז המטודה תאשר את ההגדרות של השחקן הנבחר הנוכחי.

טענת כניסה: האם הבחירה היא יציאה או כניסה

**סיבוכיות:** O(1) **שם הפונקציה:**

public void ChangeColor()  
תיאור הפונקציה: המטודה פותחת חלון לבחירת צבעים ומשנה את צבעם של כל הדמויות על המסך לצבע הנבחר.

**סיבוכיות:** O(N) כאשר N הוא מס הדמויות הנתנות לבחירה

**שם הפונקציה:**

public void randomColor()  
תיאור הפונקציה: משנה את צבעם של הדמויות לצבע רנדומאלי

**סיבוכיות:** O(N) כאשר N הוא מס הדמויות הנתנות לבחירה

**שם הפונקציה:**

public void changePlayerCPU(int n)

תיאור הפונקציה: הפעולה משנה את השחקן במקום שהתקבל מAI לאנושי או ההפך.

טענת כניסה: השחקן שרוצים לשנות לו את ההגדרה

**סיבוכיות:** O(1).

**שם הפונקציה:**

public void changePlayerAmount(int n)

תיאור הפונקציה: המטודה משנה את מס השחקנים המשתתפים במשחק.

טענת כניסה: מס השחקנים החדש שרוצים לשנות

**סיבוכיות:** O(N) כאשר N הוא מס השחקנים החדש.

**שם הפונקציה:**

private void scroll(int goPlusMin)

private void movNext(int num)

תיאור הפונקציה: הפעולות משנות את השורה והעמודה שעומדים עליה כרגע.

טענת כניסה: כמה עמודות או שורות להתקדם

**סיבוכיות:** O(1).

**13.** מחלקת לוח- Board  
public class Board

**מאפיינים:**

private Block \_board[][];

public static final int ***\_rows*** = 9, ***\_cols*** = 9;// blocks size

public final static int ***\_boardXOffset*** = *calcXOffset*(), ***\_boardYOffset*** = 40;// graphics board start offset

private ArrayList<Barrier> \_obstacles;// list of barriers in the board

**שם הפונקציה**private static final int calcXOffset()

**פונקציות:**

תיאור הפונקציה: הפונקציה מחשבת את המקום שבו האופסט של הלוח יתחיל ממנו.

טענת יציאה: תוצאות החישוב

**סיבוכיות:** O(1) **שם הפונקציה:**

public Board()  
תיאור הפונקציה: פעולה בונה, מאפסת את הלוח ואת כל המשבצות בו, חוסמת את גבולות הלוח.

**סיבוכיות:** O(N) כאשר N הוא מס המשבצות

**שם הפונקציה:**

public void setPlayerPosition(Player p, int row, int col)

public void setBarrierPosition(Barrier b, int row, int col)  
תיאור הפונקציה: הפעולות ממקמות את השחקן או המחסום במקום התחלתי על הלוח גם בצורה גרפית וגם בצורה לוגית על ידי חישוב חשבוני

טענת כניסה: מחסום/שחקן ושורה ועמודה להציב אותו

**סיבוכיות:** O(1)  **שם הפונקציה:**

public boolean canPutBarrier(Barrier b, int row, int col)

תיאור הפונקציה: הפעולה מקבלת מחסום ושורה ועמודה שרוצים למקם אותו בה, הפעולה מחזירה האם המקום הינו תקין מבחינת מחסומים אחרים.

טענת כניסה: מחסום שורה ועמודה

טענת יציאה: האם ניתן לשים את המחסום במקום המתקבל

**סיבוכיות:** O(1).

**שם הפונקציה:**

public Boolean canGoTo(int row, int col, int dir)

תיאור הפונקציה: המטודה מקבלת נקודה כולשהי על הלוח, ומחזירה האם ניתן ללכת לכיוון שהתקבל מאותה נקודה(מתייחס לשחקן).

טענת כניסה: שורה, עמודה , כיוון

טענת יציאה: האם ניתן להתקדם לכיוון הזה מהמקום המתקבל

**סיבוכיות:** O(1).

**שם הפונקציה:**

public boolean addBarrier(Barrier b)

תיאור הפונקציה: המטודה מקבלת מחסום ומוסיפה אותו במידה וזה אפשרי מבחינת הלוח.

טענת כניסה: מחסום להוספה

טענת יציאה: האם התווסף

**סיבוכיות:** O(1).

**שם הפונקציה:**

public void removeBarrier(Barrier b)

תיאור הפונקציה: הפעולה מסירה מהלוח את המחסום שהתקבל ומאפסת את החסימה שלו במשבצות.

טענת כניסה: מחסום להסיר

**סיבוכיות:** O(N) כאשר N הוא מס המחסומים.

**14.** מחלקת שחקן- Player  
public class Player extends BoardItem

**מאפיינים:**

private int \_playerNum;// the player number

private int \_steps;

private int \_speed;// how fast the x and y of the player will change

// (make the player faster or slower visuality)

private boolean \_AI = true;// whether the player is AI or not

// moving vectors

private int \_dx;// direction x

private int \_dy;// direction y

// position to reach

private int \_px;// x to reach

private int \_py;// y to reach

// the len left to move

private int \_MovingXToGo;// the x len

private int \_MovingYToGo;// the y len

private boolean \_moving = false;// is the player moving now or not

private boolean \_canPutBarrier = true;

private boolean \_settingABarrier = false;// is the player is trying to put a barrier

private Barrier \_toPut;// the player's barrier

private int \_barriersLeft;// how many barriers left in the barrier station/ the player barriers

private Point \_barrierStation;// where the visuality point of the player barriers is locate and starting from

private int movsCount;// how many moves do the player have

// when the player moving toward a cell that already have a player

// and there is a barrier behind them so the player(this) should be able to move

// again

private boolean \_finishedPlay = true;// is the player finish his turn

private Point \_goal;// x= row y= col

// the goal is where the player should reach to win

public Player(int num)

**פונקציות:**

וpublic void init()

תיאור הפונקציה: **מאפסים את השחקן ונתוניו סיבוכיות O(1)**

טענת כניסה: מס השחקן

**שם הפונקציה**public void setStation()

תיאור הפונקציה: המטודה מציבה את נקודת ההתחלה של תחנת המחסומים של השחקן.

**סיבוכיות:** O(1) **שם הפונקציה:**

public void startPlay()  
תיאור הפונקציה: הפעולה מאפסת את התור של השחקן.

**סיבוכיות:** O(1)

**שם הפונקציה:**

public void setSettingABarrier(int row, int col, boolean rotate)  
תיאור הפונקציה: הפעולה מכניסה /מוציא את השחקן ל/מ מצב בניית מחסום

טענת כניסה: שורה, עמודה והאם המחסום מסובב

**סיבוכיות:** O(1)  **שם הפונקציה:**

public boolean puttingBarrier()

תיאור הפונקציה: הפעולה מנסה למקם את המחסום במקום שהוא עומד בו כעט ומחזירה האם הוא הוא הצליח להתמקם שם.

טענת יציאה: האם ניתן להציב את המחסום

**סיבוכיות:** O(1).

**שם הפונקציה:**

public void rotateBarrier()

תיאור הפונקציה: המטודה מסובבת את המחסום שהשחקן מנסה לבנות במידה וקיים.

**סיבוכיות:** O(1).

**שם הפונקציה:**

public boolean putBarrierIn(int row, int col, boolean rotate)

תיאור הפונקציה: הפעולה יוצרת מחסום במקום ספציפי שהתקבל ויוצרת אותו גם לפי תחונת הסיבוב של המחסום.

טענת כניסה: שורה עמודה והאם מסובב

טענת יציאה: האם ניתן למקם את המחסום בנק המתקבלת

**סיבוכיות:** O(1).

**שם הפונקציה:**

public Boolean canGoTo(int x, int y, int dir)

תיאור הפונקציה: הפעולה בודקת האם שחקן יכול לזוז לנקודה כולשהי לפי התנאים של הלוח ומקום השחקן.

טענת כניסה: נק בלוח וכיוון

טענת יציאה: האם ניתן להתקדם בלוח בכיוון הזה

**סיבוכיות:** O(1).

**שם הפונקציה:**

public Boolean setMovIfPlayerInfront(Block b, int dir)

תיאור הפונקציה: פעולה רקורסיבית אשר בודקת מה לעשות כאשר קיים שחקן/שחקנים בכיוון שעליו השחקן רוצה ליפנות עליו ומציבה נתונים כמו- כמה משבצות ניתן לשחקן לדלג מעליהם או האם יש לו עוד תור מכיוון שקיים מחסום מאחורי היריב.

טענת כניסה: משבצת בלוח וכיוון להתקדם עליו

טענת יציאה: האם מאחורי השחקן/נים יש מחסום או לא

**סיבוכיות:** O(N) N מס השחקנים בכיוון שהיריב רוצה לפנות עליו.

**שם הפונקציה:**

public Boolean setDir(int d)

תיאור הפונקציה: פעולה מקבלת כיוון שעליו השחקן רוצה לפנות , אם ניתן לגשת לכיוון הזה (גם אם קיימים תנאים מיוחדים שפורטו ליעל) מקדמת אותו לפי מס הצעדים שעליו לעשות.

טענת כניסה: כיוון

טענת יציאה: האם ניתן להתקדם לכיוון

**סיבוכיות:** O(1).

**שם הפונקציה:**

private void mov(int d)

תיאור הפונקציה: פעולה המקבלת כיוון ומציבה לשחקן את המשתנים על מנת לזוז לכיוון הזה גרפית ולוגית.

טענת כניסה: כיוון להתקדם עליו

**סיבוכיות:** O(1).

**15.** מחלקת מחסום- Barrier  
ppublic class Barrier extends BoardItem

**מאפיינים:**

// the graphics sizes of the barrier

public static final int ***thickness*** = 7;// the space between two blocks

public static final int ***placeHolder*** = (Block.***size***) \* 2 + ***thickness***;// the length of the whole barrier(the part that

// should block)

private int \_blockDir;// the direction of the barrier(towards the player) which I decided it can only

// be set to up(rotate=false) or left(rotate=true)

// width and height are depend on whether the barrier is rotate or not (when not

// rotated width = placeHolder, height = thickness)

private int width;

private int height;

// the Holders are supposed to be the extended of the barrier

// (one should be 0 and the other 1)

// they are like booleans but I choosed to use them as integer for calculations

private int \_rowHolder;

private int \_colHolder;

private boolean \_rotated;// is the barrier rotated

private int \_playerNum;// the player who put the barrier's num- used inorder to pick the player color

**פונקציות:**

**שם הפונקציה**public Barrier(int row, int col, Board b, int num, boolean rotated)

תיאור הפונקציה: המטודה מאתחלת את המחסום בכיוון מסויים ובנקודה ספציפית שהתקבלה.

טענת כניסה: שורה ועמודה לוח המשחק מס שחקן והאם מסובב

**סיבוכיות:** O(1) **שם הפונקציה:**

public void rotateBarrier()  
תיאור הפונקציה: הפעולה משנה את הכיוון שעליו פונה המחסום.

**סיבוכיות:** O(1)

**שם הפונקציה:**

public Boolean setDir(int d)  
תיאור הפונקציה: הפעולה בודקת האם המחסום יכול לזוז לנקודה הזו, אם כן אז מזיזה אותו לשם, אם לא אז הפונצקיה מעבירה אותו לקיצון של הצד השני אשר רצה לפנות עליו(מלמטה למעלה מימין לשמאל והפוך)

טענת כניסה: כיוון

טענת יציאה: האם ניתן להתקדם לכיוון

**סיבוכיות:** O(1)  **שם הפונקציה:**

public void mov(int d)

תיאור הפונקציה: הפעולה מזיזה את המחסום לכיוון שהתקבל.

**סיבוכיות:** O(1).

**שם הפונקציה:**

public void rotateBarrier()

תיאור הפונקציה: המטודה מסובבת את המחסום שהשחקן מנסה לבנות במידה וקיים.

**סיבוכיות:** O(1).

**שם הפונקציה:**

public Boolean canGoTo(int x, int y)

תיאור הפונקציה: הפעולה מחזירה האם המחסום יכול לזוז בכיוון שהתקבל.

טענת כניסה: איקס ווואי

טענת יציאה: הפעולה מחזירה האם המחסום יכול לזוז בכיוון שהתקבל

**סיבוכיות:** O(1).

**16.** מחלקת הכיוונים- Directions  
public class Directions

**מאפיינים:**

public static final int ***block*** = Block.***size***;// graphic size of a block

private static int *counter* = 0;// direction counter - works as an Enum /\*

public static final int ***UP*** = *counter*++;

public static final int ***RIGHTUP*** = *counter*++;

public static final int ***RIGHT*** = *counter*++;

public static final int ***RIGHTDOWN*** = *counter*++;

public static final int ***DOWN*** = *counter*++;

public static final int ***DOWNLEFT*** = *counter*++;

public static final int ***LEFT*** = *counter*++;

public static final int ***LEFTUP*** = *counter*++;

public static final int ***NumDirs*** = *counter*;

// \*/ Enums of directions

// arrays of the physic moving

// last cells-0,0 is for same place

public static final int ***Dirsx***[] = { 0, 1, 1, 1, 0, -1, -1, -1, 0 };// x/cols dir

public static final int ***Dirsy***[] = { -1, -1, 0, 1, 1, 1, 0, -1, 0 };// y/rows dir

**פונקציות:**

**שם הפונקציה**public static int fromPointDir(int r1, int c1, int r2, int c2)

תיאור הפונקציה: המטודה מחזירה את הכיוון מנקודה של שורה ועמודה 1 לשורה ועמודה 2.

טענת כניסה: נקודה אחת ונקודה שניה (נקודה=שורה ועמודה)

טענת יציאה: הכיוון מהנקודה הראשונה לשניה

**סיבוכיות:** O(1) **שם הפונקציה:**

public static int oppositeDir(int d)  
הפעולה מחזירה כיוון הפוך לכיוון שהתקבל.

טענת כניסה: כיוון

טענת יציאה: הכיוון ההפוך

**סיבוכיות:** O(1)

## .15**תוכנית ראשית**

package Main;

import java.awt.Dimension;

import java.awt.Toolkit;

import javax.swing.JFrame;

public class Main {

public static void main(String[] args) {

// screen size

Dimension DimMax = Toolkit.getDefaultToolkit().getScreenSize();

// jframe settings

JFrame window = new JFrame("Quoridor");

window.setUndecorated(true);

window.setResizable(false);

// set full screen

window.setMinimumSize(DimMax);

window.setExtendedState(JFrame.MAXIMIZED\_BOTH);

window.setLocationRelativeTo(null);

window.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

window.add(new GamePanel(DimMax.width, DimMax.height));

window.pack();

window.setVisible(true);

}

}

## **16. מדריך למשתמש**

**1 הוראות התקנה**

יש להעתיק את התיקייה Quoridor למקום כלשהו במחשב.

**2 הוראות הפעלה**

יש להיכנס לתיקייה Quoridor וללחוץ דאבל קליק על קובץ המשחק Quoridor.jar.

במשחק ישנם שני דרכי הפעלה במחשב: שימוש במקלדת ושימוש בלחיצות העכבר.

\*כל הפתורים בצבע שחור נתונים ללחיצה בכל רגע נתון\*

במסך הכניסה: זזים עם החצים למעלה ולמטה ובוחרים באנטר

במסך ההוראות: ניתן רק ללחוץ על הכפתורים עם העכבר- קליק ימני

במסך ההגדרות: ניתן לבחור שחקן בשני דרכים

דרך 1:זזים עם החצים (למעלה, למטה,ימינה,שמאלה) ובוחרים את השחקן שרוצים באנטר

דרך 2:לוחצים עם העכבר קליק ימני על שחקן שרוצים לבחורולוחצים עליו שנית כדי לאשר אותו(כמו ללחוץ על אנטר)

על מנת לבטל בחירה של הגדרה של שחקן לוחצים על הכפתור ESC אם הכפתור נלחץ ולא הוגדר עדיין שחקן אחד, הלחיצה תעביר למסך התפריט.

ניתן ללחוץ על הכפתורים שחורים עם העכבר

הכפתור שרשום עליו color משנה את צבע השחקנים שבוחרים כעט

כפתור שרשום עליו + מוסיף עוד שחקן כפתור – מוריד שחקן

כפתורים שרשום עליהם CPU ו Player הם מייצגים האם השחקן שהכפתור מוצג ליידו הוא שחקן ממוחשב או אנושי, ניתן לשנות זאת אחד מן השני בלחיצה על הכפתור (לכל השחקנים כל הזמן גם אם בוחרים הגדרות של שחקן אחר)

כפתור היציא כדי לצאת לתפריט

במסך המשחק: על מנת לזוז עם השחקן או עם המחסום, משתמשים בחצים(לכיוון הרצוי)

על מנת ליצור מחסום אנו לוחצים על המקש B על מנת לסובב את המחסום אנו לוחצים על המקש R , על מנת למקם את המחסום אנחנו לוחצים על אנטר.

ניתן ללחוץ על הכפתור EXIT כדי לצאת.

בסיום המשחק שמופעים הודעות הנצחון מופעים שני כפתורים

שניתן ללחוץ עליהם

לחיצה על Restart מאתחל את המשחק עם אותן דמויות והגדרות מחדש.

לחיצה על Back תחזיר למסך ההגדרות.

**3 הודעות מערכת**

ישנה הודעה בסיום המשחק בהתאם לניצחון\הפסד השחקן האנושי.

**מסך ניצחון:**

**שחקן 1:**

**שחקן 2:**



**שחקן 3:**



**שחקן 4:**



## **17. קוד הפרויקט**

מחלקת Block

package Board;

public class Block {

//the graphics size of a block

public static final int ***size*** = 45;

//booleans flag whether the direction(by its name) is blocked

private Boolean \_up;

private Boolean \_down;

private Boolean \_left;

private Boolean \_right;

//Temporary there can be two players in a block

private int \_player;

//init all sides to be unblocked

public Block() {

\_up = \_down = \_left = \_right = false;

\_player=0;

}

public void setBlocked(int dir) {

// input- the direction the player wants to block

// output- void

if (dir == Directions.***UP***) {

set\_up(true);

}

if (dir == Directions.***DOWN***) {

set\_down(true);

}

if (dir == Directions.***RIGHT***) {

set\_right(true);

}

if (dir == Directions.***LEFT***) {

set\_left(true);

}

}

public void removeBlocked(int dir) {

// input- the direction to unblock

// output- void

if (dir == Directions.***UP***) {

set\_up(false);

}

if (dir == Directions.***DOWN***) {

set\_down(false);

}

if (dir == Directions.***RIGHT***) {

set\_right(false);

}

if (dir == Directions.***LEFT***) {

set\_left(false);

}

}

public Boolean get\_up() {

return \_up;

}

private void set\_up(boolean b) {

this.\_up = b;

}

public Boolean get\_down() {

return \_down;

}

private void set\_down(boolean b) {

this.\_down = b;

}

public Boolean get\_left() {

return \_left;

}

private void set\_left(boolean b) {

this.\_left = b;

}

public Boolean get\_right() {

return \_right;

}

private void set\_right(boolean b) {

this.\_right = b;

}

public int get\_player() {

return \_player;

}

public void set\_player(Boolean flag) {

\_player+=flag?1:-1;

if(\_player<0)

\_player=0;

}

}

מחלקת Board

package Board;

import java.util.ArrayList;

import Entity.Barrier;

import Entity.Player;

import Main.GamePanel;

public class Board {

private Block \_board[][];

public static final int \_rows = 9, \_cols = 9;// blocks size

public final static int \_boardXOffset = calcXOffset(), \_boardYOffset = 40;// graphics board start offset

private ArrayList<Barrier> \_obstacles;// list of barriers in the board

private static final int calcXOffset() {

// calculate the offset of the board to make the board be in the screen half

int res = GamePanel.WIDTH / 2 - (\_cols / 2 + \_cols % 2) \* (Block.size + Barrier.thickness)

+ (Block.size + Barrier.thickness) / 2;

return res;

}

public Board() {

\_obstacles = new ArrayList<Barrier>();

\_board = new Block[\_rows][\_cols];

// init rows

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

// init cols

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

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

}

}

// blocking the out of Board/Range areas

// Horizontal

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

\_board[0][i].setBlocked(Directions.UP);

\_board[\_rows - 1][i].setBlocked(Directions.DOWN);

}

// Vertical

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

\_board[i][0].setBlocked(Directions.LEFT);

\_board[i][\_cols - 1].setBlocked(Directions.RIGHT);

}

}

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

return \_board;

}

public ArrayList<Barrier> get\_obstacles() {

return \_obstacles;

}

public Block getBlock(int row, int col) {

// input- the row and col of a block

// output- the block itself

if (row >= 0 && row < \_rows && col >= 0 && col < \_cols)

return \_board[row][col];

// the block isnt in the range

return null;

}

public void setPlayerPosition(Player p, int row, int col) {

// input- the player and the start position to set him in the board

// the function set his row and col and his graphics x and y

// output- void

if (row >= 0 && row < \_rows && col >= 0 && col < \_cols) {

p.setRowCol(row, col);

// calc player position

int x = \_boardXOffset + (col) \* (Block.size + Barrier.thickness);

int y = \_boardYOffset + (row) \* (Block.size + Barrier.thickness);

p.setPosition(x, y);

\_board[row][col].set\_player(true);

}

}

public void setBarrierPosition(Barrier b, int row, int col) {

// input- the barrier and the first position to set him in the board

// the function set his row and col and his graphics x and y

// output- void

if (row >= 0 && row < \_rows && col >= 0 && col < \_cols) {

b.setRowCol(row, col);

int x = \_boardXOffset + (col) \* (Block.size + Barrier.thickness) - Barrier.thickness \* b.get\_rowHolder();

int y = \_boardYOffset + (row) \* (Block.size + Barrier.thickness) - Barrier.thickness \* b.get\_colHolder();

b.setPosition(x, y);

}

}

public boolean canPutBarrier(Barrier b, int row, int col) {

// input- a barrier , row and col we want to set it

// output- is it possible to put the barrier in this position

// \*the barrier direction=d

// if the position out of index or the d side in the square is blocked (there is

// already a barrier) then it means the position is unavailable

return !(col < 0 || row < 0 || col >= \_cols || row >= \_rows

|| getBlock(row, col).get\_up() && b.getDir() == Directions.UP

|| getBlock(row, col).get\_down() && b.getDir() == Directions.DOWN

|| getBlock(row, col).get\_right() && b.getDir() == Directions.RIGHT

|| getBlock(row, col).get\_left() && b.getDir() == Directions.LEFT);

}

public Boolean canGoTo(int row, int col, int dir) {

// input- row and col (of the player) and a direction the player wants to move

// output- if the player can move to this direction(the way doesnt blocked)

int x = Directions.Dirsx[dir], y = Directions.Dirsy[dir];

Block b = getBlock(row + y, col + x);

boolean res = false;

if (b != null) {

if (dir == Directions.UP)

res = !b.get\_down();

else if (dir == Directions.DOWN)

res = !b.get\_up();

else if (dir == Directions.RIGHT)

res = !b.get\_left();

else if (dir == Directions.LEFT)

res = !b.get\_right();

}

return res;

}

public boolean addBarrier(Barrier b) {

// input- a barrier that the player wants to add

// output- is it possible to put it ( add it to the barrier list and block the

// surround area of the barrier)

int dir = b.getDir();

// first

int row = b.get\_row(), col = b.get\_col();

// second part

int colH = b.get\_colHolder(), rowH = b.get\_rowHolder();

// check the position of both part of the barrier if its legal position the

// place it

if (canPutBarrier(b, row, col) && canPutBarrier(b, row + rowH, col + colH)) {

// first pos

\_board[row][col].setBlocked(dir);

\_board[row + rowH][col + colH].setBlocked(dir);

// second pos

dir = Directions.oppositeDir(dir);

\_board[row - colH][col - rowH].setBlocked(dir);

\_board[row + rowH - colH][col + colH - rowH].setBlocked(dir);

\_obstacles.add(b);

return true;

}

return false;

}

public void removeBarrier(Barrier b) {

// input- barrier to remove

// output- void

// the function remove the barrier if the list contains it

if (\_obstacles.contains(b)) {

int dir = b.getDir();

// first

int row = b.get\_row(), col = b.get\_col();

// second part

int colH = b.get\_colHolder(), rowH = b.get\_rowHolder();

// first pos

\_board[row][col].removeBlocked(dir);

\_board[row + rowH][col + colH].removeBlocked(dir);

// second pos

dir = Directions.oppositeDir(dir);

\_board[row - colH][col - rowH].removeBlocked(dir);

\_board[row + rowH - colH][col + colH - rowH].removeBlocked(dir);

\_obstacles.remove(b);

}

}

}

מחקלת Directions

package Board;

public class Directions {

public static final int ***block*** = Block.***size***;// graphic size of a block

private static int *counter* = 0;// direction counter - works as an Enum /\*

public static final int ***UP*** = *counter*++;

public static final int ***RIGHTUP*** = *counter*++;

public static final int ***RIGHT*** = *counter*++;

public static final int ***RIGHTDOWN*** = *counter*++;

public static final int ***DOWN*** = *counter*++;

public static final int ***DOWNLEFT*** = *counter*++;

public static final int ***LEFT*** = *counter*++;

public static final int ***LEFTUP*** = *counter*++;

public static final int ***NumDirs*** = *counter*;

// \*/ Enums of directions

// arrays of the physic moving

// last cells-0,0 is for same place

public static final int ***Dirsx***[] = { 0, 1, 1, 1, 0, -1, -1, -1, 0 };// x/cols dir

public static final int ***Dirsy***[] = { -1, -1, 0, 1, 1, 1, 0, -1, 0 };// y/rows dir

public static int fromPointDir(int r1, int c1, int r2, int c2) {

// input- to positions in the board

// output- the direction to reach the second position from the first one \*only

// works horizontal and vertical not between (no need between)

int r = r1 - r2;

int c = c1 - c2;

if (r > 0)

return ***UP***;

else if (r < 0)

return ***DOWN***;

else if (c > 0)

return ***LEFT***;

else

return ***RIGHT***;

}

public static int oppositeDir(int d) {

//input- a direction

//output- the opposite direction

if (d == ***UP***)

return ***DOWN***;

if (d == ***DOWN***)

return ***UP***;

if (d == ***RIGHT***)

return ***LEFT***;

if (d == ***LEFT***)

return ***RIGHT***;

return -1;

}

}

מחלקת Barrier

package Entity;

import Board.Block;

import Board.Board;

import Board.Directions;

public class Barrier extends BoardItem {

// the graphics sizes of the barrier

public static final int thickness = 7;// the space between two blocks

public static final int placeHolder = (Block.size) \* 2 + thickness;// the length of the whole barrier(the part that

// should block)

private int \_blockDir;// the direction of the barrier(towards the player) which I decided it can only

// be set to up(rotate=false) or left(rotate=true)

// width and height are depend on whether the barrier is rotate or not (when not

// rotated width = placeHolder, height = thickness)

private int width;

private int height;

// the Holders are supposed to be the extended of the barrier

// (one should be 0 and the other 1)

// they are like booleans but I choosed to use them as integer for calculations

private int \_rowHolder;

private int \_colHolder;

private boolean \_rotated;// is the barrier rotated

private int \_playerNum;// the player who put the barrier's num- used inorder to pick the player color

public Barrier(int row, int col, Board b, int num, boolean rotated) {

// input- the barrier information

\_board = b;

// fix vertical limts

if (!rotated && row == 0)

row++;

if (!rotated && col + 1 >= \_board.\_cols)

col -= 1;

// fix horizontal limits

if (rotated && row == Board.\_rows - 1)

row--;

if (rotated && col == 0)

col++;

\_rotated = rotated;

// set the information according to the rotating

// \*explanation is in the variables statement\*

if (!\_rotated) {

\_blockDir = Directions.UP;

width = placeHolder;

height = thickness;

\_rowHolder = 0;

\_colHolder = 1;

} else {

\_blockDir = Directions.LEFT;

width = thickness;

height = placeHolder;

\_rowHolder = 1;

\_colHolder = 0;

}

\_playerNum = num;

b.setBarrierPosition(this, row, col);

}

public int getDir() {

return \_blockDir;

}

public int get\_playerNum() {

return \_playerNum;

}

public void rotateBarrier() {

// the function purpose is to liken a barrier rotation by changing it variables

// the variables values are swapping and keeping the symmetry

\_rotated = !\_rotated;

\_rowHolder += \_colHolder;

\_colHolder = \_rowHolder - \_colHolder;

\_rowHolder -= \_colHolder;

width += height;

height = width - height;

width -= height;

if (\_rotated) {

\_x -= thickness;

\_y += thickness;

\_blockDir = Directions.LEFT;

}

else {

\_x += thickness;

\_y -= thickness;

\_blockDir = Directions.UP;

}

// out of bound fixes

if (!\_rotated && \_row <= 0)

mov(Directions.DOWN);

if (\_rotated && \_row + 1 >= Board.\_rows)

mov(Directions.UP);

if (\_rotated && \_col == 0)

mov(Directions.RIGHT);

if (!\_rotated && \_col + 1 >= Board.\_cols)

mov(Directions.LEFT);

}

public Boolean canGoTo(int x, int y) {

// input- x direction and y direction

// output- whether the barrier can move to this position (\_row + y, \_col + x)

Block b = \_board.getBlock(\_row + y, \_col + x);

Block b1 = \_board.getBlock(\_row + y + \_rowHolder, \_col + x + \_colHolder);

if (b == null || b1 == null || \_rotated && \_col + x == 0 || !\_rotated && \_row + y == 0)

return false;

return true;

}

public Boolean setDir(int d) {

// input- direction that we want to move the barrier to

// output- moving the barrier to this direction

// or if the barrier is in the first/last index of the row/col

// so the function localize the barrier in the end/start of the row/col

// (from end to start or reverse)

// the function doesnt in charge to change the row col x y ,

// only to manage this situation and change the moving direction

// or keep it the same and let it change location

int len = 1;

if (!canGoTo(Directions.Dirsx[d], Directions.Dirsy[d])) {

d = Directions.oppositeDir(d);

len = \_board.\_rows - 2;

}

for (int i = 0; i < len; i++)

mov(d);

return true;

}

public void mov(int d) {

// input- the direction to move the barrier to

//outcome- the values updated according to the move

int x = Directions.Dirsx[d], y = Directions.Dirsy[d];

\_x += x \* (Directions.block + thickness);

\_y += y \* (Directions.block + thickness);

\_row += y;

\_col += x;

}

public int getWidth() {

return width;

}

public int getHeight() {

return height;

}

public int get\_rowHolder() {

return \_rowHolder;

}

public int get\_colHolder() {

return \_colHolder;

}

}

מחלקת BoardItem

package Entity;

import Board.Board;

public abstract class BoardItem {

// a class of a board item

// contains basic variables that a board item should have

protected Board \_board;// the board that the item is in

protected int \_col;

protected int \_row;

protected int \_x;

protected int \_y;

public void setPosition(int x, int y) {

\_x = x;

\_y = y;

}

public void setRowCol(int row, int col) {

this.\_row = row;

this.\_col = col;

}

public int get\_col() {

return \_col;

}

public int get\_row() {

return \_row;

}

public int get\_x() {

return \_x;

}

public int get\_y() {

return \_y;

}

}

מחלקת Player

package Entity;

import java.awt.Point;

import Board.Block;

import Board.Board;

import Board.Directions;

import Handlers.Turns;

public class Player extends BoardItem {

private int \_playerNum;// the player number

private int \_steps;

private int \_speed;// how fast the x and y of the player will change

// (make the player faster or slower visuality)

private boolean \_AI = true;// whether the player is AI or not

// moving vectors

private int \_dx;// direction x

private int \_dy;// direction y

// position to reach

private int \_px;// x to reach

private int \_py;// y to reach

// the len left to move

private int \_MovingXToGo;// the x len

private int \_MovingYToGo;// the y len

private boolean \_moving = false;// is the player moving now or not

private boolean \_canPutBarrier = true;

private boolean \_settingABarrier = false;// is the player is trying to put a barrier

private Barrier \_toPut;// the player's barrier

private int \_barriersLeft;// how many barriers left in the barrier station/ the player barriers

private Point \_barrierStation;// where the visuality point of the player barriers is locate and starting from

private int movsCount;// how many moves do the player have

// when the player moving toward a cell that already have a player

// and there is a barrier behind them so the player(this) should be able to move

// again

private boolean \_finishedPlay = true;// is the player finish his turn

private Point \_goal;// x= row y= col

// the goal is where the player should reach to win

public Player(int num) {

// input- the player number

// init turn/act variables

init();

// init settings variables

\_playerNum = num;

\_speed = 3;

\_barrierStation = new Point();

setStation();

}

public void init() {

\_barriersLeft = 10;

\_MovingXToGo = 0;

\_MovingYToGo = 0;

\_canPutBarrier = true;

}

// set the position of the station according to the player num

public void setStation() {

int x = \_board.\_boardXOffset, y = \_board.\_boardYOffset;

int height, width;

height = \_board.\_rows \* (Barrier.thickness + Block.size);

width = \_board.\_cols \* (Barrier.thickness + Block.size);

if (\_playerNum == 1)

\_barrierStation.setLocation(x + width / 3.5, y - 105);

else if (\_playerNum == 2) {

\_barrierStation.setLocation(x + width / 3.5, y + height);

} else if (\_playerNum == 3) {

\_barrierStation.setLocation(x - 105, y + height / 3.5);

} else {

\_barrierStation.setLocation(x + width, y + height / 3.5);

}

}

public Point get\_barrierStation() {

return \_barrierStation;

}

public int get\_barriersLeft() {

return \_barriersLeft;

}

public void addBarrierLeft() {

\_barriersLeft++;

}

public void setBoard(Board b) {

\_board = b;

}

public void startPlay() {

// init variables for a new turn

\_canPutBarrier = true;

\_finishedPlay = false;

movsCount = 1;

Turns.setCurrAction(1);

}

public int getPlayerNum() {

return \_playerNum;

}

public void setGoal(int row, int col) {

\_goal = new Point(row, col);

}

public Point getGoal() {

return \_goal;

}

public void setSettingABarrier(int row, int col, boolean rotate) {

// input- where to init the barrier at and is it rotated

// outcome- the player will have a barrier in the board which he will try to put

// or he wont have it anymore (depend on his previous choice)

// the function switching between put/barrier mode

// on= try to place a barrier

// off= not trying

// if the player doesnt have barriers left

// or if the player is stadning in other player block

// that means he cant try to put a barrier

if ((\_barriersLeft > 0) && \_canPutBarrier) {

\_settingABarrier = !\_settingABarrier;

// if the is now trying to set a new barrier we need to create a barrier to put

if (\_settingABarrier)

\_toPut = new Barrier(row, col, \_board, \_playerNum, rotate);

else// else the player doesnt need a barrier

\_toPut = null;

}

}

public void setSettingABarrier(boolean rotate) {

// set the barrier start position in the player current location

setSettingABarrier(\_row, \_col, rotate);

}

public boolean puttingBarrier() {

// output- did the barrier Place successfully

boolean put = false;

if (\_settingABarrier && \_board.addBarrier(\_toPut)) {

// if the player is in put mode and the barrier successfully placed

// make the barrier dissapear/exit barrier mode

setSettingABarrier(false);

\_finishedPlay = true;

Turns.setCurrAction(2);

\_barriersLeft--;

put = true;

}

return put;

}

public void rotateBarrier() {

// rotate the barrier

if (\_settingABarrier)

\_toPut.rotateBarrier();

}

public boolean putBarrierIn(int row, int col, boolean rotate) {

// input- specific location to place a barrier and its rotation

boolean put = false;

setSettingABarrier(row, col, rotate);// Switch into put mode

if (\_settingABarrier)

put = puttingBarrier();

return put;

}

public Barrier get\_toPut() {

return \_toPut;

}

public boolean is\_settingABarrier() {

return \_settingABarrier;

}

public Boolean setMovIfPlayerInfront(Block b, int dir) {

// recursion function that handle a situation where the player want to go to a

// direction and there is/are player in the direction

// (also handle the situation where there is a barrier behind them)

// if there is no player in this block that mean the way after the player(s) is

// free

if (b.get\_player() == 0)

return true;

// if there is a block in the way that mean

if (dir == Directions.UP && b.get\_up() || dir == Directions.DOWN && b.get\_down()

|| dir == Directions.LEFT && b.get\_left() || dir == Directions.RIGHT && b.get\_right()) {

movsCount = 2;

\_canPutBarrier = false;

mov(dir);

return false;

}

\_steps++;

int x = Directions.Dirsx[dir];

int y = Directions.Dirsy[dir];

return setMovIfPlayerInfront(\_board.getBlock(\_row + y \* \_steps, \_col + x \* \_steps), dir);

}

public boolean is\_moving() {

return \_moving;

}

public boolean isAI() {

return \_AI;

}

public void setAI(boolean \_AI) {

this.\_AI = \_AI;

}

public boolean isFinishedPlay() {

return \_finishedPlay;

}

public Boolean canGoTo(int x, int y, int dir) {

// input- direction and the x and y of the direction

// output- whether the player can move to this dir or not

boolean res = false;

\_steps = 1;

Block b = \_board.getBlock(\_row + y, \_col + x);

if (b != null) {

if (dir == Directions.UP)

res = !b.get\_down();

else if (dir == Directions.DOWN)

res = !b.get\_up();

else if (dir == Directions.RIGHT)

res = !b.get\_left();

else if (dir == Directions.LEFT)

res = !b.get\_right();

if (res && b.get\_player() > 0)

res = setMovIfPlayerInfront(b, dir);

}

return res;

}

public Boolean setDir(int d) {

// input- direction that the player wants to go/move the barrier to

// output- is the player / the barrier moving

// the function proceed the move only if its leagal

boolean res = false;

if (!\_moving) {

if (\_settingABarrier) {

res = \_toPut.setDir(d);

} else if (canGoTo(Directions.Dirsx[d], Directions.Dirsy[d], d)) {

mov(d);

res = true;

}

}

return res;

}

private void mov(int d) {

// input- direction to move to

// the function proceed the move even if its ileagal

// the function change the player info according to the move

int x = Directions.Dirsx[d], y = Directions.Dirsy[d];

\_dx = \_speed \* x;

\_dy = \_speed \* y;

\_MovingXToGo = x \* (Directions.block + Barrier.thickness) \* \_steps;

\_MovingYToGo = y \* (Directions.block + Barrier.thickness) \* \_steps;

\_px = \_x + \_MovingXToGo;

\_py = \_y + \_MovingYToGo;

\_MovingXToGo = Math.abs(\_MovingXToGo);

\_MovingYToGo = Math.abs(\_MovingYToGo);

\_moving = true;

\_board.get\_board()[\_row][\_col].set\_player(false);

\_row += y \* \_steps;

\_col += x \* \_steps;

if (\_row >= 0 && \_col >= 0 && \_row < Board.\_rows && \_col < Board.\_cols)

\_board.get\_board()[\_row][\_col].set\_player(true);

}

public boolean isWin() {

// output- is the player in the row/col of their goal

return \_goal.x == \_row || \_goal.y == \_col;

}

public void update() {

// updateing the player position to make it looks like he acctualy moving

if (\_moving) {

if (\_MovingXToGo > 0) {

\_x += \_dx;

\_MovingXToGo -= Math.abs(\_dx);

}

if (\_MovingYToGo > 0) {

\_y += \_dy;

\_MovingYToGo -= Math.abs(\_dy);

}

// when the player is finished moving the turn move to the next player

// he finish moving when he doesnt have anymore x and y to go

if (\_MovingXToGo <= 0 && \_MovingYToGo <= 0) {

\_moving = false;

\_dx = \_dy = 0;

\_x = \_px;

\_y = \_py;

Turns.setCurrAction(1);

movsCount--;

if (movsCount == 0)

\_finishedPlay = true;

}

}

}

}

מחלקת GameBoard

package GameState;

import java.awt.Graphics2D;

import java.awt.event.MouseEvent;

import java.util.Random;

import Board.Board;

import Board.Directions;

import Entity.Barrier;

import Graphics.Background;

import Graphics.Graphics;

import Graphics.Img;

import Graphics.PopMessage;

import Handlers.Keys;

import Handlers.Players;

import Handlers.Turns;

import Logic.AI;

import Logic.Graph;

import Logic.LegalChecks;

import Main.GamePanel;

public class GameBoard extends GameState {

private Background \_bg;// board background

private Board \_board;// the actual game board

private Graph \_graph;// the graph of the board

private int \_playerWon = 0;// 0 = havent win yet , other= the winner

private Img \_exitButton;// button to exit the game

private boolean pause = false;// pause

// pop messages

private PopMessage \_restart;

private PopMessage \_changeSettings;

private PopMessage \_playerWonMessage;

private PopMessage \_playerWonNumerMessage;// init to null then a player win the image change

private PopMessage \_wonMessage;

public GameBoard(GameStateManager gsm) {

this.\_gsm = gsm;

init();

}

@Override

public void init() {

// background

\_bg = new Background("/Backgrounds/board.png");

// winning buttons and UI

\_changeSettings = new PopMessage("/buttons/Win/Back.png", 0 - 258, GamePanel.HEIGHT / 2 + 90, 220, 0, 1, 0);

\_restart = new PopMessage("/buttons/Win/Restart.png", GamePanel.WIDTH, GamePanel.HEIGHT / 8 + 120, 250, 0, -1,

0);

\_playerWonMessage = new PopMessage("/buttons/Win/PlayerWon.png", -200, GamePanel.HEIGHT / 10,

GamePanel.WIDTH / 45, 0, 5, 0);

\_playerWonNumerMessage = null;

\_wonMessage = new PopMessage("/buttons/Win/Won.png", GamePanel.WIDTH - 320, 0 - 360, 0, GamePanel.HEIGHT / 9, 0,

2);

// init board and graph

\_board = new Board();

\_graph = new Graph(\_board);

AI.initBoard(\_board);

// init players location

for (int i = 0; i < Players.\_players.length; i++)

Players.\_players[i].setBoard(\_board);

setPlayersPositions();

LegalChecks.init(Players.\_players, \_graph);

\_exitButton = new Img("/buttons/settings/exit.png", GamePanel.WIDTH - 100, GamePanel.HEIGHT - 90, 90, 13);

\_playerWon = 0;

// current turn set -1 in order to start need to set it to next turn (0)

nextTurn();

}

public void setPlayersPositions() {

// init the play positions to their start and their goal to the opposite side

int x1 = 0, y1 = \_board.\_cols / 2, x2 = \_board.\_rows - 1, y2 = \_board.\_cols / 2, x3 = \_board.\_rows / 2, y3 = 0,

x4 = \_board.\_rows / 2, y4 = \_board.\_cols - 1;

// -1 mean the whole col or row

\_board.setPlayerPosition(Players.\_players[0], x1, y1);

Players.\_players[0].setGoal(x2, -1);

\_board.setPlayerPosition(Players.\_players[1], x2, y2);

Players.\_players[1].setGoal(x1, -1);

if (Turns.numTurns >= 3) {

\_board.setPlayerPosition(Players.\_players[2], x3, y3);

Players.\_players[2].setGoal(-1, y4);

}

if (Turns.numTurns >= 4) {

\_board.setPlayerPosition(Players.\_players[3], x4, y4);

Players.\_players[3].setGoal(-1, y3);

}

}

@Override

public void update() {

// pause

if (Keys.isPressed(Keys.ESCAPE))

pause = !pause;

if (pause)

return;

// handle game management if the game havent finish

if (\_playerWon == 0) {

// can play when the previous player finished playing

if (Players.\_players[Turns.getPrevTurn()].isFinishedPlay()) {

// if the current palyer isnt ai so the game is waiting for the player input

// and checking if the player finish his turn

if (!Turns.getCurrTurnPlayer().isAI()) {

if (Turns.getCurrTurnPlayer().isFinishedPlay())

nextTurn();

// if the player isnt moving he should input the move

else if (!Turns.getCurrTurnPlayer().is\_moving())

handleInput();

}

// AI turn

else {

// update the AI

AI.update();

if (AI.isFinishedTurn())

nextTurn();

}

}

}

// update all players

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

Players.\_players[i].update();

// checking if the player is won

if (Players.\_players[i].isFinishedPlay() && Players.\_players[i].isWin()) {

Won(i);

}

}

}

private void Won(int i) {

// input- the winning player index

// outcome- pop message of the winning player and show the popping messages

\_playerWon = i + 1;

if (\_playerWonNumerMessage == null)

\_playerWonNumerMessage = new PopMessage(Graphics.numbers[i], GamePanel.WIDTH + 3, GamePanel.HEIGHT / 10,

GamePanel.WIDTH / 33, 0, -5, 0);

\_playerWonMessage.update();

if (\_playerWonMessage.is\_isFinishMoving())

\_playerWonNumerMessage.update();

if (\_playerWonNumerMessage.is\_isFinishMoving())

\_wonMessage.update();

if (\_wonMessage.is\_isFinishMoving()) {

\_changeSettings.update();

\_restart.update();

}

}

@Override

public void draw(Graphics2D g) {

// input- Graphics variable

// outcome- the function is drawing the whole game

// background

\_bg.draw(g);

// board

Graphics.drawBoard(g, \_board.get\_board().length, \_board.get\_board()[0].length);

// ui

if (Turns.\_currentTurn != -1)

Graphics.drawCurrPlayerTurn(g);

\_exitButton.drawImg(g);

// players draws

int len = Players.\_players.length;

for (int i = 0; i < len; i++) {

// barrier if player is bulding

if (Players.\_players[i].is\_settingABarrier())

Graphics.drawBarierr(g, Players.\_players[i].get\_toPut().get\_x(),

Players.\_players[i].get\_toPut().get\_y(), Players.\_players[i].get\_toPut().getWidth(),

Players.\_players[i].get\_toPut().getHeight(), i + 1);

// player

Graphics.drawPlayer(Players.\_players[i].get\_x(), Players.\_players[i].get\_y(), i, g);

}

// draw the barrier stations

for (int i = 0; i < 2; i++) {

Graphics.drawPlayerBarriers(g, Players.\_players[i].get\_barriersLeft(),

Players.\_players[i].get\_barrierStation(), 20, 0, Barrier.thickness, Barrier.placeHolder, i + 1);

}

for (int i = 2; i < len; i++) {

Graphics.drawPlayerBarriers(g, Players.\_players[i].get\_barriersLeft(),

Players.\_players[i].get\_barrierStation(), 0, 20, Barrier.placeHolder, Barrier.thickness, i + 1);

}

// Barriers on the board

Graphics.drawBarriers(\_board.get\_obstacles(), g);

Graphics.drawCurrPlayerUI(g);

// player won

if (\_playerWon != 0) {

\_playerWonMessage.draw(g);

\_wonMessage.draw(g);

if (\_playerWonNumerMessage != null)

\_playerWonNumerMessage.draw(g);

\_changeSettings.draw(g);

\_restart.draw(g);

}

}

public boolean handleLogic() {

// output- is the previous act was legal

boolean legal = true;

if (Turns.\_curAction == 2) {

// check if the last barrier that was placed position's is legal

Barrier b = \_board.get\_obstacles().get(\_board.get\_obstacles().size() - 1);

\_graph.initBarrier(b);// init the graph

if (!LegalChecks.barrierPositionLegal()) {

// the act was ileagal

// need to take of the barrier and "give it back to the player"

\_board.removeBarrier(b);

\_graph.initBarrier(b);

Turns.getCurrTurnPlayer().addBarrierLeft();

Turns.getCurrTurnPlayer().startPlay();// init the player turn

AI.init();// init the AI and let it know that a barrier was try to be puted

AI.tryToPutBarrier();

legal = false;

}

}

return legal;

}

public boolean nextTurn() {

// set to the next turn only if the last action was legal

// output- did the turn moved to the next player

boolean res = false;

if (handleLogic()) {

Turns.nextTurn();

res = true;

}

return res;

}

@Override

public void handleInput() {

// if player trying to put barrier so the game wait untill he press enter and

// puts it

// else the game check if the player pressed any key

int cur = Turns.\_currentTurn;

// change to barrier mode/move mode

if (Keys.isPressed(Keys.BB))

Players.\_players[cur].setSettingABarrier((new Random()).nextBoolean());

// rotate the barrier

if (Keys.isPressed(Keys.RB))

Players.\_players[cur].rotateBarrier();

if (Keys.isPressed(Keys.ENTER))

Players.\_players[cur].puttingBarrier();

// move player/barrier

if (Keys.isPressed(Keys.UP))

Players.\_players[cur].setDir(Directions.UP);

if (Keys.isPressed(Keys.RIGHT))

Players.\_players[cur].setDir(Directions.RIGHT);

if (Keys.isPressed(Keys.DOWN))

Players.\_players[cur].setDir(Directions.DOWN);

if (Keys.isPressed(Keys.LEFT))

Players.\_players[cur].setDir(Directions.LEFT);

}

@Override

public void MouseClicked(MouseEvent e) {

// is clicked the exit button then exit

if (\_exitButton.isClicked(e))

\_gsm.setState(\_gsm.MENUSTATE);

// if the player won and clicked on one of the pop messages then

if (\_playerWon != 0) {

// restart the game

if (\_restart.isClicked(e)) {

Players.initPlayers();

AI.init();

Turns.init();

init();

}

// exit back to settings

if (\_changeSettings.isClicked(e))

\_gsm.setState(\_gsm.SETTINGS);

}

}

}

מחלקת GameState

package GameState;

import java.awt.Graphics2D;

import java.awt.event.MouseEvent;

public abstract class GameState {

//abstract class ,simulate a game page

protected GameStateManager \_gsm;//the page manger

public abstract void init();//init page

public abstract void update();//update page

public abstract void draw(Graphics2D g);//draw page

public abstract void handleInput();//listen to input in page

public abstract void MouseClicked(MouseEvent e);//listen to mouse clicks in page

}

מחלקת GameStateManager

package GameState;

import java.awt.Graphics2D;

import java.awt.event.MouseEvent;

import Graphics.Background;

public class GameStateManager {

// game states array

private GameState[] \_gameStates;

// the index of the current state/page

private int \_currentState;

// load screen image

public Background \_loadScreen;

// enum of game states

public static int \_statCounter = 0;

public static final int MENUSTATE = \_statCounter++;

public static final int SETTINGS = \_statCounter++;

public static final int GAMESTATE = \_statCounter++;

public static final int HELPSTATE = \_statCounter++;

public static final int NUMGAMESTATES = \_statCounter;

public GameStateManager() {

// init the game pages array

\_gameStates = new GameState[NUMGAMESTATES];

\_loadScreen = new Background("/Backgrounds/load.gif");

\_currentState = MENUSTATE;

loadState(\_currentState);

}

private void loadState(int state) {

// input- a page num from the enum

// outcome- load the page according to the enum

if (state == MENUSTATE)

\_gameStates[state] = new MenuState(this);

else if (state == SETTINGS)

\_gameStates[state] = new SettingsState(this);

else if (state == GAMESTATE)

\_gameStates[state] = new GameBoard(this);

else if (state == HELPSTATE)

\_gameStates[state] = new HelpState(this);

}

private void unloadState(int state) {

// input- page from the dnum

// unload the given page

\_gameStates[state] = null;

}

public void setState(int state) {

// input- a page num from the enum

// outcome- unload the current page and load the given one

unloadState(\_currentState);

\_currentState = state;

loadState(\_currentState);

}

public void update() {

// update the current page

if (\_gameStates[\_currentState] != null)

\_gameStates[\_currentState].update();

}

public void draw(Graphics2D g) {

// draw the current page

// if it havent load then draw the loading screen

if (\_gameStates[\_currentState] != null)

\_gameStates[\_currentState].draw(g);

else {

\_loadScreen.draw(g);

}

}

public void MouseClicked(MouseEvent e) {

// handle mouse click in the page

\_gameStates[\_currentState].MouseClicked(e);

}

}

מחלקת HelpState

package GameState;

import java.awt.Graphics2D;

import java.awt.event.MouseEvent;

import Graphics.Background;

import Graphics.Img;

import Main.GamePanel;

public class HelpState extends GameState {

//buttons

private Img next;

private Img prev;

private Img \_exitButton;

//the backgrounds are used as a pages of information that the knowledge the player have is drawn/ written in the actual picture

private Background pages[];

private int currentPage;//used to swap between the background- like swapping pages

public HelpState(GameStateManager gsm) {

this.\_gsm = gsm;

init();

}

@Override

public void init() {

//init the backgrounds, there are 4 images and their names are 0 to 3 with the same ending -jpg

pages=new Background[4];

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

pages[i]=new Background("/Backgrounds/Help/"+(i)+".jpg");

}

//init buttons

next=new Img("/buttons/help/next.png", GamePanel.WIDTH - 100, GamePanel.HEIGHT - 90, 90, 13);

prev=new Img("/buttons/help/prev.png", 10, GamePanel.HEIGHT - 90, 90, 13);

\_exitButton = new Img("/buttons/settings/exit.png", 10, 30, 90, 13);

//init to the first page

currentPage=0;

}

@Override

public void update() {

}

@Override

public void draw(Graphics2D g) {

//draw all the buttons and the background

pages[currentPage].draw(g);

g.drawString(String.valueOf(currentPage), GamePanel.WIDTH/2, 10);

next.drawImg(g);

prev.drawImg(g);

\_exitButton.drawImg(g);

}

@Override

public void handleInput() {

}

@Override

public void MouseClicked(MouseEvent e) {

//if the next button was pressed then switch to the next page,if the previous button was pressed switch to the previous

if(next.isClicked(e)&&currentPage<pages.length-1)

currentPage++;

if(prev.isClicked(e)&&currentPage>0)

currentPage--;

if (\_exitButton.isClicked(e))

\_gsm.setState(\_gsm.MENUSTATE);

}

}

מחלקת MenuState

package GameState;

import java.awt.\*;

import java.awt.event.MouseEvent;

import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;

import Graphics.Background;

import Handlers.Animation;

import Handlers.Keys;

import Main.GamePanel;

public class MenuState extends GameState {

private Background \_bg;// the menu background

private BufferedImage \_button;// a button image

private BufferedImage \_selcted;// selected button image

private Animation \_glow;// glow effect/ animation

private BufferedImage \_recB;// rectangle button

private int \_currentChoice = 0;// the current button choice

private String[] \_options = { "Start", "Help", "Quit" };// buttons strings

private Font \_font;

public MenuState(GameStateManager gsm) {

// input- the main page manger

this.\_gsm = gsm;

init();

}

public void init() {

// init all the variables

try {

\_bg = new Background("/Backgrounds/menu.png");

\_button = ImageIO.read(getClass().getResourceAsStream("/buttons/Title/btn.png"));

\_selcted = ImageIO.read(getClass().getResourceAsStream("/buttons/Title/Selected.png"));

\_recB = ImageIO.read(getClass().getResourceAsStream("/buttons/Title/recB.png"));

\_font = new Font("Arial", Font.PLAIN, 12);

BufferedImage spritesheet = ImageIO.read(getClass().getResourceAsStream("/buttons/Title/glow.png"));

BufferedImage glow[] = new BufferedImage[8];

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

glow[i] = spritesheet.getSubimage(i \* 50, 0, 50, 50);

\_glow = new Animation();

\_glow.setDelay(150);

\_glow.setFrames(glow);

\_glow.set\_revloop(true);

} catch (Exception e) {

e.printStackTrace();

}

}

public void update() {

\_glow.update();// update the glow animation

handleInput();// check keys

}

public void draw(Graphics2D g) {

// draw the page

// draw bg

\_bg.draw(g);

// draw menu options

g.setFont(\_font);

g.setColor(Color.BLACK);

int x = 10, y = GamePanel.HEIGHT / 2;

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

g.drawImage(\_recB, x, y + 30 \* i, 20, 20, null);

if (i == \_currentChoice) {

g.drawImage(\_selcted, x, y + 30 \* i, GamePanel.WIDTH - x, 20, null);

g.drawImage(\_glow.getImage(), x, y + 30 \* i, 20, 20, null);

} else {

g.drawImage(\_button, x, y + 30 \* i, GamePanel.WIDTH - x, 20, null);

}

g.drawString(\_options[i], x + 30, y + 15 + 30 \* i);

}

}

private void select() {

// start settings page

if (\_currentChoice == 0) {

\_gsm.setState(GameStateManager.SETTINGS);

}

if (\_currentChoice == 1) {

// help page

\_gsm.setState(GameStateManager.HELPSTATE);

}

// exit game

if (\_currentChoice == 2) {

System.exit(0);

}

}

@Override

public void handleInput() {

//init animation

Boolean glowreset = false;

//pressed enter so need to make a selection

if (Keys.isPressed(Keys.ENTER))

select();

//swapping to the next option

if (Keys.isPressed(Keys.UP)) {

if (\_currentChoice > 0)

\_currentChoice--;

else

\_currentChoice = \_options.length - 1;

glowreset = true;

}

if (Keys.isPressed(Keys.DOWN)) {

if (\_currentChoice < \_options.length - 1)

\_currentChoice++;

else

\_currentChoice = 0;

glowreset = true;

}

//rest animation

if (glowreset)

\_glow.init();

}

@Override

public void MouseClicked(MouseEvent e) {

//none

}

}

מחלקת SettingsState

package GameState;

import java.awt.Color;

import java.awt.Font;

import java.awt.Graphics2D;

import java.awt.event.MouseEvent;

import java.util.Random;

import Graphics.Background;

import Graphics.Graphics;

import Graphics.Img;

import Graphics.Sprites;

import Handlers.ColorPicker;

import Handlers.Keys;

import Handlers.Players;

import Handlers.Turns;

import Logic.AI;

import Main.GamePanel;

public class SettingsState extends GameState {

// images of all the optional players\sprites to play as

private Img \_sprites[][];

private ColorPicker \_cp;// tool for choosing a color

// imgs size

private final int imgBoxWidth = 30;

private final int imgBoxHeight = 51;

// current row and col of player

private int \_currentRow = 0;

private int \_currentCol = 0;

// current players in the game

private int \_numOfPlayers = 2;

// the player who we set their settings for

private int \_currPlayerSettings = 1;

private boolean canStart = false;// if we set the settings for all the player we can start the game

// images and graphics

private Font \_font;

private Font \_bigFont;

private Img \_exitButton;

private Img \_AddRemovePlayer[];

private Img \_AI[];

private Img \_red;

private Img \_black;

private Img \_changeColor;

private Background \_bg;

public SettingsState(GameStateManager gsm) {

this.\_gsm = gsm;

init();

}

@Override

public void init() {

// init background and buttons

\_bg = new Background("/Backgrounds/sttings.jpg");

\_sprites = Sprites.load("/Sprites/sprites.png", imgBoxWidth, imgBoxHeight, 100, 5);

\_red = new Img("/Backgrounds/red.png", 0, 0, 30, 45);

\_black = new Img("/Backgrounds/black.png", 0, 0, 30, 45);

\_changeColor = new Img("/buttons/settings/color.png", GamePanel.WIDTH - 130, 20, 90, 13);

\_exitButton = new Img("/buttons/settings/exit.png", GamePanel.WIDTH - 100, GamePanel.HEIGHT - 60, 90, 13);

// init all the handlers that we will use during the game and set their options

// in this page

Turns.init();

AI.init();

Players.init(\_numOfPlayers);

Graphics.setPlayers();

\_font = new Font("Arial", Font.PLAIN, 10);

\_bigFont = new Font("Arial", Font.BOLD, 100);

\_AddRemovePlayer = new Img[2];

\_AddRemovePlayer[0] = new Img("/buttons/settings/Plus.png", 0, 0, 35, 26);

\_AddRemovePlayer[1] = new Img("/buttons/settings/Minus.png", 0, 0, 35, 26);

\_AI = new Img[4];

\_AI[0] = new Img("/buttons/settings/Player.png", 0, 0, 73, 13);// player on set to Human at start

// (can be changed)

\_AI[1] = new Img("/buttons/settings/CPU.png", 0, 0, 73, 13);

\_AI[2] = new Img("/buttons/settings/CPU.png", 0, 0, 73, 13);

\_AI[3] = new Img("/buttons/settings/CPU.png", 0, 0, 73, 13);

\_cp = new ColorPicker();

randomColor();// set random color at start

// player 1 is init to be human player

changePlayerCPU(0);

}

@Override

public void update() {

handleInput();

}

@Override

public void draw(Graphics2D g) {

\_bg.draw(g);

Font currentFont = g.getFont();

g.setFont(\_font);

\_changeColor.drawImg(g);

\_exitButton.drawImg(g);

int x = 100, y;

g.setColor(Color.black);

// drawing the sprites and a box behind them

// (red = current row and col, black = other)

for (int i = 0; i < \_sprites[0].length - 1; i++) {

y = 5;

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

if (!canStart && j == \_currentRow && i == \_currentCol)

\_red.drawImg(g, x, y);

else

\_black.drawImg(g, x, y);

\_sprites[j][i].drawImg(g);

y += 50;

}

x += 50;

}

// draw the level row bar

int height = 500;

int point = (int) (height / (\_sprites.length - 1)) \* \_currentRow;

point = (point == 0) ? 5 : point;

g.fillRect(40, 5, 30, height);

g.setColor(Color.red);

g.fillRect(40, point, 30, 10);

// settings

x += 390;

y = 50;

// draw the choosed sprites of the players

for (int i = 0; i < \_numOfPlayers; i++) {

y += 50;

\_black.drawImg(g, x, y);

// draw the current row col sprite and mark the current player who choosing his

// player

if (i + 1 == \_currPlayerSettings) {

\_red.drawImg(g, x, y);

\_sprites[\_currentRow][\_currentCol].drawImg(g, x, y);

}

g.drawImage(Graphics.player[i], x, y, null);

g.drawString("Player: " + String.valueOf(i + 1).toString(), x + 40, y + 22);

\_AI[i].setImgCords(x - 90, y);

\_AI[i].drawImg(g);

}

// draw the minus button only if there are more than the minimum players

if (\_numOfPlayers != 2) {

\_AddRemovePlayer[1].setImgCords(x + 50, y + 60);

\_AddRemovePlayer[1].drawImg(g);

}

// draw the plus button only if there are less than the maximum players

if (\_numOfPlayers != 4) {

\_AddRemovePlayer[0].setImgCords(x, y + 60);

\_AddRemovePlayer[0].drawImg(g);

}

// start UI

if (canStart) {

g.setFont(\_bigFont);

int w = 300;

int h = 100;

x = GamePanel.WIDTH / 2 - w / 2;

y = GamePanel.HEIGHT / 2 - h;

g.setColor(Color.red);

g.fillRoundRect(x, y, w, h, 30, 30);

g.setColor(Color.white);

g.drawString("Start?", x + 10, y + 80);

}

// back the the default font

g.setFont(currentFont);

}

private void select(boolean exit) {

// input- boolean that flag is the selection was to cancel

// outcome- set player settings or cancel them according to exit

// if the first player didnt set the screen set back to the menu

Random rnd = new Random();

// start choose sprite

// canel options

if (exit) {

// get back to menu

if (\_currPlayerSettings == 1)

\_gsm.setState(GameStateManager.MENUSTATE);

else {

// set the previus player settings

\_currPlayerSettings -= 1;

Graphics.setSprite(null, \_currPlayerSettings);

canStart = false;

}

} // enter opptions

else if (canStart) {

// if the game can start then set the numer of turns and move to the board panel

Turns.setTurns(\_numOfPlayers);

\_gsm.setState(GameStateManager.GAMESTATE);

} else {

// set the current player's settings

// set their sprite

Graphics.setSprite(Sprites.cutBufferedImage(\_sprites[\_currentRow][\_currentCol].getImage()),

\_currPlayerSettings);

// move to the next player to set their settings

\_currPlayerSettings++;

// random color

randomColor();

// random position

int scrolls = rnd.nextInt(8) \* 2;

int movenext = rnd.nextInt(3) \* 2;

for (int i = 0; i < scrolls; i++)

scroll(1);

for (int i = 0; i < movenext; i++)

movNext(1);

// if the maximum of player that set to play in the game settings were set then

// that means we can start the game now

if (\_currPlayerSettings > \_numOfPlayers)

canStart = true;

}

}

public void ChangeColor() {

// open a tool kit that let the player choose their sprite color

Color c = \_cp.ColorChoose();

Graphics.setColor(c, \_currPlayerSettings);// set the player color in the graphics

// load the sprites again so the color will be easyer to change

\_sprites = Sprites.load("/Sprites/sprites.png", imgBoxWidth, imgBoxHeight, 100, 5);

// set the choosed color to all the sprites

for (int i = 0; i < \_sprites[0].length - 1; i++) {

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

\_sprites[j][i].setImg(Sprites.setImgColor(c, \_sprites[j][i].getImage()));

}

}

}

public void randomColor() {

// set random color to the sprites

Color c = Sprites.randomColor();// get a random color

Graphics.setColor(c, \_currPlayerSettings);// set the player color in the graphics

// load the sprites again so the color will be easyer to change

\_sprites = Sprites.load("/Sprites/sprites.png", imgBoxWidth, imgBoxHeight, 100, 5);

// set the choosed color to all the sprites

for (int i = 0; i < \_sprites[0].length - 1; i++) {

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

\_sprites[j][i].setImg(Sprites.setImgColor(c, \_sprites[j][i].getImage()));

}

}

}

public void changePlayerCPU(int n) {

// input- number of player that we want to set as an AI

Players.\_players[n].setAI(!Players.\_players[n].isAI());

// change the picture

if (Players.\_players[n].isAI())

\_AI[n].setImg("/buttons/settings/CPU.png");

else

\_AI[n].setImg("/buttons/settings/Player.png");

}

public void changePlayerAmount(int n) {

// input- 1 or 0, 1= add 0=remove;

int amount = \_numOfPlayers;

if (n == 0) {

amount++;

canStart = false;

} else {

amount--;

if (\_currPlayerSettings >= \_numOfPlayers)

\_currPlayerSettings -= 1;

}

// add or remove in the player class a player

if (Players.addOrRemovePlayer(amount)) {

\_numOfPlayers = amount;

Graphics.addOrRemovePlayer(amount);

}

}

@Override

public void MouseClicked(MouseEvent e) {

// check the input

// AI change

for (int i = 0; i < \_numOfPlayers; i++) {

if (\_AI[i].isClicked(e)) {

changePlayerCPU(i);

}

}

// add or remove player

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

if (\_AddRemovePlayer[i].isClicked(e)) {

changePlayerAmount(i);

}

}

// color changer

if (\_changeColor.isClicked(e))

ChangeColor();

else if (\_exitButton.isClicked(e))

\_gsm.setState(\_gsm.MENUSTATE);// exit to menu

else {

// pressed on sprite

// if so then set the current row col to the sprite

for (int i = 0; i < \_sprites[0].length - 1; i++)

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

if (\_sprites[j][i].isClicked(e)) {

if (\_currentRow == j && \_currentCol == i) {

select(false);

} else {

\_currentRow = j;

\_currentCol = i;

}

}

}

}

private void scroll(int goPlusMin) {

//input- direction of the row to go up=-1 down=1

//init to the end or the start if the row get out of range

\_currentRow += goPlusMin;

// up

if (\_currentRow < 0) {

\_currentRow = \_sprites.length + \_currentRow;

}

// down

if (\_currentRow > \_sprites.length - 1) {

\_currentRow = \_currentRow - \_sprites.length;

}

}

private void movNext(int num) {

//input- direction of the col to go up=-1 down=1

//init to the end or the start if the col get out of range

\_currentCol += num;

if (\_currentCol < 0)

\_currentCol = \_sprites[0].length - 2;

if (\_currentCol >= \_sprites[0].length - 1)

\_currentCol = 0;

}

@Override

public void handleInput() {

//handel input

if (Keys.isPressed(Keys.ENTER))

select(false);

if (Keys.isPressed(Keys.ESCAPE))

select(true);

if (!canStart) {

if (Keys.isPressed(Keys.UP)) {

scroll(-1);

}

if (Keys.isPressed(Keys.DOWN)) {

scroll(1);

}

if (Keys.isPressed(Keys.LEFT)) {

movNext(-1);

}

if (Keys.isPressed(Keys.RIGHT)) {

movNext(1);

}

}

}

}

מחלקת BackGround

package Graphics;

import Main.GamePanel;

import java.awt.\*;

public class Background {

//image of the background

private Img \_image;

public Background(String s) {

try {

//set the image to the size of the screen

\_image=new Img(s, 0, 0, GamePanel.WIDTH, GamePanel.HEIGHT);

} catch (Exception e) {

e.printStackTrace();

}

}

public void draw(Graphics2D g) {

//draw it

\_image.drawImg(g);

}

}

מחלקת Graphics

package Graphics;

import java.awt.Color;

import java.awt.Graphics2D;

import java.awt.Point;

import java.awt.image.BufferedImage;

import java.util.ArrayList;

import Board.Block;

import Board.Board;

import Entity.Barrier;

import Entity.Player;

import Handlers.Players;

import Handlers.Turns;

import Main.GamePanel;

public class Graphics {

// image of a block

public static final Img \_block = new Img("/Board/block.png", 1, 1, Block.size, Block.size);

// players sprites

public static BufferedImage player[];

// the size of player

private static int \_playSize = Block.size;

public static String numbers[];// path to numbers imges

// players colors

public static Color \_playerColor[];

public static void setPlayers() {

// set the arrays according to the number of players

player = new BufferedImage[Players.\_players.length];

\_playerColor = new Color[Players.\_players.length];

numbers = new String[4];

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

numbers[i] = "/buttons/Win/" + (i + 1) + ".png";

}

public static void drawCurrPlayerTurn(Graphics2D g) {

// drawing who is the current player on the screen

int num = Turns.\_currentTurn + 1;

String str = "Player " + num + " Turn";

g.setColor(Color.black);

g.fillRect(GamePanel.WIDTH - 90, 10, 75, 15);

g.setColor(\_playerColor[num - 1]);

g.drawString(str, GamePanel.WIDTH - 90, 20);

}

public static void drawCurrPlayerUI(Graphics2D g) {

// draw above the current player/their barrier his number

Player p = Turns.getCurrTurnPlayer();

if (p != null) {

int x = p.get\_x(), y = p.get\_y() - 10;

if (p.is\_settingABarrier()) {

x = p.get\_toPut().get\_x();

y = p.get\_toPut().get\_y() - 10;

}

int num = Turns.\_currentTurn + 1;

String str = "Player " + num;

g.setColor(Color.black);

g.fillRect(x, y - 10, 45, 15);

g.setColor(\_playerColor[num - 1]);

g.drawString(str, x, y);

}

}

public static void drawPlayer(int x, int y, int i, Graphics2D g) {

// draw given player in given position

if (player[i] != null)

g.drawImage(player[i], x, y, \_playSize, \_playSize, null);

}

public static void drawPlayerBarriers(Graphics2D g, int count, Point p, int dx, int dy, int width, int height,

int playerNum) {

// draw player's barrier station

int x = p.x, y = p.y;

for (int i = 0; i < count; i++) {

drawBarierr(g, x, y, width, height, playerNum);

x += dx;

y += dy;

}

}

public static void drawBlock(Graphics2D g, int x, int y) {

// draw a block

\_block.drawImg(g, x, y);

}

public static void drawBarierr(Graphics2D g, int x, int y, int width, int height, int num) {

// draw a barrier

// (can be player's and board's

g.setColor(\_playerColor[num - 1]);

g.fillRect(x, y, width, height);

}

public static void drawBarriers(ArrayList<Barrier> obstacles, Graphics2D g) {

// draw the barriers from the array list

for (int i = 0; i < obstacles.size(); i++) {

int x = obstacles.get(i).get\_x();

int y = obstacles.get(i).get\_y();

int width = obstacles.get(i).getWidth();

int height = obstacles.get(i).getHeight();

drawBarierr(g, x, y, width, height, obstacles.get(i).get\_playerNum());

}

}

public static void drawBoard(Graphics2D g, int col, int row) {

// draw all the blocks of the board

g.setColor(Color.black);

int x = Board.\_boardXOffset, y;

for (int i = 0; i < col; i++) {

y = Board.\_boardYOffset;

for (int j = 0; j < row; j++) {

drawBlock(g, x, y);

y += Barrier.thickness + Block.size;

}

x += Barrier.thickness + Block.size;

}

}

// set Charecter Colors

public static void setSprite(BufferedImage img, int i) {

player[i - 1] = img;

}

// set color of player in i

public static void setColor(Color c, int i) {

if (i <= \_playerColor.length)

\_playerColor[i - 1] = c;

}

public static boolean addOrRemovePlayer(int amount) {

//change the amount of players and their graphics data

if (amount > 4 || amount <= 1)

return false;

int len = player.length;

if (amount < len)

len = amount;

Color c[] = new Color[amount];

BufferedImage p[] = new BufferedImage[amount];

for (int i = 0; i <= len - 1; i++) {

p[i] = player[i];

c[i] = \_playerColor[i];

}

player = p;

\_playerColor = c;

return true;

}

}

מחלקת Img

package Graphics;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.event.MouseEvent;

import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;

import Main.GamePanel;

public class Img {

//image

private BufferedImage \_image;

//acord and size

private int x, y, width, height;

public Img(String s, int x, int y, int width, int height) {

//init image to the size and location

try {

\_image = ImageIO.read(getClass().getResourceAsStream(s));

} catch (Exception e) {

e.printStackTrace();

}

setImgCords(x, y);

setImgSize(width, height);

}

public Img(BufferedImage img, int x, int y, int width, int height) {

//init image size and location

\_image = img;

setImgCords(x, y);

setImgSize(width, height);

}

public void drawImg(Graphics g) {

//Drawing the image according to its info

Graphics2D g2d = (Graphics2D) g;

g2d.drawImage(\_image, x, y, width, height, null);

}

public void drawImg(Graphics g, int x1, int y1) {

//Drawing the image in given location

Graphics2D g2d = (Graphics2D) g;

g2d.drawImage(\_image, x1, y1, width, height, null);

}

public void setImgCords(int x, int y) {

this.x = x;

this.y = y;

}

public void setImgSize(int width, int height) {

this.width = width;

this.height = height;

}

public void setImg(BufferedImage image) {

\_image = image;

}

public void setImg(String s) {

try {

\_image = ImageIO.read(getClass().getResourceAsStream(s));

} catch (Exception e) {

e.printStackTrace();

}

}

public BufferedImage getImage() {

return \_image;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public int getWidth() {

return width;

}

public int getHeight() {

return height;

}

public boolean isClicked(MouseEvent e) {

int ex = e.getX() / GamePanel.SCALE, ey = e.getY() / GamePanel.SCALE;

return this.x <= ex && this.x + this.width >= ex && this.y <= ey && this.y + this.height >= ey;

}

}

מחלק PopMessage

package Graphics;

import java.awt.Graphics;

import java.awt.event.MouseEvent;

public class PopMessage {

// pop message that show on screen

// the picture

private Img \_pic;

// accords

private int \_x;

private int \_y;

// max len/distanse of moving to go

private int \_maxY;

private int \_maxX;

// vectors of direction

// Default speed is 3

private int \_dx = 3;

private int \_dy = 3;

// size

private int \_width;

private int \_hight;

// when finishing to move

private boolean \_isFinishMoving;

public PopMessage(String path, int x, int y, int maxX, int maxY, int xdir, int ydir) {

// input- info of the message

// output- create

\_maxX = maxX;

\_maxY = maxY;

\_x = x;

\_y = y;

\_dx \*= xdir;

\_dy \*= ydir;

\_pic = new Img(path, \_x, \_y, 1, 1);

\_hight = \_pic.getImage().getHeight();

\_width = \_pic.getImage().getWidth();

\_pic.setImgSize(\_width / 2, \_hight / 2);

\_isFinishMoving = false;

}

public void draw(Graphics g) {

// draw the image

\_pic.drawImg(g);

}

public void update() {

// update the image to move if it havent finished

if (!\_isFinishMoving) {

//more x to go

if (\_maxX > 0) {

\_maxX--;

\_x += \_dx;

\_pic.setImgCords(\_x, \_y);

}

//more y to go

if (\_maxY > 0) {

\_maxY--;

\_y += \_dy;

\_pic.setImgCords(\_x, \_y);

}

//finished

if (\_maxX == 0 && \_maxY == 0)

\_isFinishMoving = true;

}

}

public boolean is\_isFinishMoving() {

return \_isFinishMoving;

}

public void set\_pic(String path) {

this.\_pic.setImg(path);

}

public boolean isClicked(MouseEvent e) {

return \_pic.isClicked(e);

}

}

מחלקת Sprites

package Graphics;

import java.awt.Color;

import java.awt.image.BufferedImage;

import java.util.Random;

import javax.imageio.ImageIO;

public class Sprites {

// set Charecter Colors

public static BufferedImage setImgColor(Color c, BufferedImage player) {

// input- image and a color the change it to

// output- the image when its color set to the given color

// the color set according to shades of white from the original picture

int w = player.getWidth();

int h = player.getHeight();

for (int i = 0; i < w; i++) {

for (int j = 0; j < h; j++) {

// go throh each pixel of the image and change its shade according to the color

// and the original picture

int cur = player.getRGB(i, j);

if (!isTransparent(i, j, player) && !isColor(i, j, player, Color.BLACK)) {

player.setRGB(i, j, setColor(cur, c));

}

}

}

return player;

}

private static int setColor(int cur, Color c) {

// input- the current RGB of the pixel and a color to change to a shade of

// output- the RGB of the mixed shade

if (c != null) {

if (cur == -1118482)

cur = c.getRGB();

else if (cur == -1)

cur = c.brighter().getRGB();

else if (cur == 16777215)

cur = c.darker().getRGB();

else if (cur == -2368549)

cur = c.brighter().darker().getRGB();

else if (cur == -3552823)

cur = c.darker().getRGB();

else if (cur == -7039852)

cur = c.brighter().brighter().getRGB();

else if (cur == -5390378)

cur = c.brighter().brighter().getRGB();

}

return cur;

}

public static Color randomColor() {

// output- random color

Random rand = new Random();

float r = rand.nextFloat();

float g = rand.nextFloat();

float b = rand.nextFloat();

return new Color(r, g, b);

}

public static boolean isTransparent(int x, int y, BufferedImage img) {

// input- position of pixel in the image img

// output- is the pixel is transparent

boolean flag = false;

int pixel = img.getRGB(x, y);

if ((pixel >> 24) == 0x00) {

flag = true;

}

return flag;

}

public static boolean isColor(int x, int y, BufferedImage img, Color c) {

// input- position in the img and a color

// output- is the pixel color is the given color

int pixel = img.getRGB(x, y);

return (pixel == c.getRGB());

}

public static Img[][] load(String s, int w, int h, int x, int y) {

// input- path to an image size of each selection in the image and position

// where the selection start from

// output- image matrix of all the selections in the image

Img[][] ret;

int tempY = y;

try {

BufferedImage spritesheet = ImageIO.read(Sprites.class.getResourceAsStream(s));

int width = spritesheet.getWidth() / w;

int height = spritesheet.getHeight() / h;

ret = new Img[height][width];

for (int j = 0; j < width; j++) {

y = tempY;

for (int i = 0; i < height; i++) {

ret[i][j] = new Img(spritesheet.getSubimage(j \* w, i \* h, w, h), x, y, w, h);

y += 50;

}

x += 50;

}

return ret;

} catch (Exception e) {

e.printStackTrace();

System.out.println("Error loading graphics.");

System.exit(0);

}

return null;

}

public static BufferedImage cutBufferedImage(BufferedImage img) {

// input- an image

// output- the same img but sliced in a way that the image size is in the size

// of the figure in the image

// cut off all the sides transparent of the figure

int w = img.getWidth();

int h = img.getHeight();

int upH = -1;

int downH = h;

// height

for (int i = 0; i < w; i++) {

int numhu = 0;

int numhd = h;

boolean swich = false;

for (int j = 0; j < h; j++) {

boolean isT = isTransparent(i, j, img);

if (isT) {

if (!swich)

numhu++;

else

numhd--;

} else

swich = true;

}

if (upH < 0)

upH = numhu;

else

upH = Math.min(upH, numhu);

if (downH >= h)

downH = numhd;

else

downH = Math.max(downH, numhd);

}

img = img.getSubimage(0, upH, w, downH - upH);

h = downH - upH;

// width

int leftW = -1, rightW = -1;

for (int j = 0; j < h; j++) {

int numhL = 0;

int numhR = w;

boolean swich = false;

for (int i = 0; i < w; i++) {

boolean isT = isTransparent(i, j, img);

if (isT) {

if (!swich)

numhL++;

else

numhR--;

} else

swich = true;

}

if (leftW < 0)

leftW = numhL;

else

leftW = Math.min(leftW, numhL);

if (rightW >= h)

rightW = numhR;

else

rightW = Math.max(rightW, numhR);

}

img = img.getSubimage(leftW, 0, rightW - leftW, h);

return img;

}

}

מחלקת Animation

package Handlers;

import java.awt.image.BufferedImage;

public class Animation {

//not a big part of the game only used for an feature in the menu

//meant to be use in the previous project

//image array

private BufferedImage[] \_frames;

//current frame

private int \_currentFrame;

private long \_startTime;

private long \_delay;

private int \_loop;

private boolean \_playedOnce;

private boolean \_revloop;

private int \_dir = 1;

public Animation() {

init();

}

public void init() {

\_playedOnce = false;

\_dir = 1;

\_currentFrame = 0;

}

public void setFrames(BufferedImage[] frames) {

//set the frame images

this.\_frames = frames;

\_currentFrame = 0;

\_startTime = System.*nanoTime*();

\_playedOnce = false;

\_loop = -1;

}

public void setDelay(long d) {

\_delay = d;

}

public void setFrame(int i) {

\_currentFrame = i;

}

public void update() {

if (\_revloop)

revLoopUpdate();

else if (\_loop == -1)

basicUpdate();

else

loopUpdate();

}

public void basicUpdate() {

if (\_playedOnce) {

return;

}

long elapsed = (System.*nanoTime*() - \_startTime) / 1000000;

if (elapsed > \_delay) {

\_currentFrame++;

\_startTime = System.*nanoTime*();

}

if (\_currentFrame >= \_frames.length - 1) {

\_playedOnce = true;

}

}

public void loopUpdate() {

long elapsed = (System.*nanoTime*() - \_startTime) / 1000000;

if (elapsed > \_delay) {

\_currentFrame++;

\_startTime = System.*nanoTime*();

}

if (\_currentFrame == \_frames.length) {

\_currentFrame = \_loop;

\_playedOnce = true;

}

}

public void revLoopUpdate() {

long elapsed = (System.*nanoTime*() - \_startTime) / 1000000;

if (elapsed > \_delay) {

\_currentFrame+=\_dir;

\_startTime = System.*nanoTime*();

}

if(\_playedOnce&&\_currentFrame==0) {

\_playedOnce=false;

\_dir=-\_dir;

}

if (\_currentFrame == \_frames.length-1) {

\_playedOnce = true;

\_dir=-\_dir;

}

//not overflow

if(\_currentFrame<0)

\_currentFrame=0;

if(\_currentFrame>=\_frames.length)

\_currentFrame = \_frames.length-1;

}

public void set\_revloop(boolean \_revloop) {

this.\_revloop = \_revloop;

}

public int getFrame() {

return \_currentFrame;

}

public BufferedImage getImage() {

return \_frames[\_currentFrame];

}

public void setImage(BufferedImage b) {

\_frames[\_currentFrame] = b;

}

public boolean hasPlayedOnce() {

return \_playedOnce;

}

}

מחלקת ColorPicker

package Handlers;

import java.awt.Color;

import java.awt.FlowLayout;

import javax.swing.\*;

import Graphics.Sprites;

public class ColorPicker extends JFrame {

//class of a jframe to choose a color

private Color c;

public ColorPicker() {

//init the jframe and a random color

this.setLayout(new FlowLayout());

this.pack();

c = Sprites.randomColor();

}

public Color ColorChoose() {

//choose a color and return it

c = JColorChooser.showDialog(null, "Pick a color...I guess", c);

return c;

}

public Color getColor() {

//output- the last choosed color

return c;

}

}

מחלקת Keys

package Handlers;

import java.awt.event.KeyEvent;

// this class contains a boolean array of current/previus game keys statues

// for the keys that are used for this game.

// R button key is press when keyState[k] is true and false when released or havent pressed etc.

public class Keys {

//my game buttons enum

private static int *\_counter* = 0;

public static int *UP* = *\_counter*++;

public static int *LEFT* = *\_counter*++;

public static int *DOWN* = *\_counter*++;

public static int *RIGHT* = *\_counter*++;

public static int *RB* = *\_counter*++;

public static int *BB* = *\_counter*++;

public static int *ENTER* = *\_counter*++;

public static int *ESCAPE* = *\_counter*++;

//the number of keys in my game

public static final int ***NUM\_KEYS*** = *\_counter*;

public static boolean *keyState*[] = new boolean[***NUM\_KEYS***];

public static boolean *prevKeyState*[] = new boolean[***NUM\_KEYS***];

public static void keySet(int i, boolean b) {

// vk=hash code of the key after \_

if (i == KeyEvent.***VK\_UP***)

*keyState*[*UP*] = b;

else if (i == KeyEvent.***VK\_LEFT***)

*keyState*[*LEFT*] = b;

else if (i == KeyEvent.***VK\_DOWN***)

*keyState*[*DOWN*] = b;

else if (i == KeyEvent.***VK\_RIGHT***)

*keyState*[*RIGHT*] = b;

else if (i == KeyEvent.***VK\_R***)

*keyState*[*RB*] = b;

else if (i == KeyEvent.***VK\_B***)

*keyState*[*BB*] = b;

else if (i == KeyEvent.***VK\_ENTER***)

*keyState*[*ENTER*] = b;

else if (i == KeyEvent.***VK\_ESCAPE***)

*keyState*[*ESCAPE*] = b;

}

public static void update() {

//update the previous pressed button

for (int i = 0; i < ***NUM\_KEYS***; i++) {

*prevKeyState*[i] = *keyState*[i];

}

}

public static boolean isPressed(int i) {

//input- enum key

//output- was it pressed

return *keyState*[i] && !*prevKeyState*[i];

}

}

מחלקת Players

package Handlers;

import Entity.Player;

public class Players {

// game players class

// array of the players

public static Player *\_players*[];

public static void init(int num) {

// input- num of players in the game

// init the array and players states

*\_players* = new Player[num];

for (int i = 0; i < *\_players*.length; i++)

*\_players*[i] = new Player(i + 1);

}

public static void initPlayers() {

// init players

for (int i = 0; i < *\_players*.length; i++)

*\_players*[i].init();

}

public static void setAI(int num, boolean flag) {

// input- a player num and a flag if he is an AI or not

// the function set him to AI/human according to the flag

*\_players*[num].setAI(flag);

}

public static boolean addOrRemovePlayer(int amount) {

// input- players amount

// output- if the players amount is leagal then change the players array size

// and keep the players data unless we want to remove then the one that we

// remove

boolean flag = false;

if (!(amount > 4 || amount <= 1)) {

int len = *\_players*.length;

if (amount < len)

len = amount;

Player p[] = new Player[amount];

for (int i = 0; i <= len - 1; i++)

p[i] = *\_players*[i];

for (int i = len; i < amount; i++)

p[i] = new Player(i + 1);

*\_players* = p;

flag = true;

}

return flag;

}

}

מחלקת Turns

package Handlers;

import Entity.Player;

import Logic.AI;

public class Turns {

public static int \_currentTurn=-1;//-1 means that the game is done

public static int numTurns;//num of turns

private static Player \_currentPlayer;//the current player

public static int \_curAction=0;//0- none, 1-moved, 2-barrier

public static int getNextTurn() {

//get the index of the player in the next turn

return (\_currentTurn+1<numTurns)?\_currentTurn+1:0;

}

public static int getPrevTurn() {

//get the index of the player in the previous turn

return (\_currentTurn-1>=0)?\_currentTurn-1:numTurns-1;

}

public static void setCurrAction(int n) {

//input- number of the last action that were take

\_curAction=n;

}

public static void nextTurn() {

//set data to the next turn

\_curAction=0;

\_currentTurn=getNextTurn();

\_currentPlayer=Players.\_players[\_currentTurn];

\_currentPlayer.startPlay();//start player turn

AI.startThink();//init AI

}

public static void init() {

\_currentTurn=-1;

\_currentPlayer = null;

}

public static void setTurns(int \_numOfPlayers) {

numTurns=\_numOfPlayers;

}

public static Player getCurrTurnPlayer() {

return \_currentPlayer;

}

}

מחלקת AI

package Logic;

import java.util.ArrayList;

import java.util.Random;

import Board.Board;

import Board.Directions;

import Entity.Barrier;

import Entity.Player;

import Handlers.Players;

import Handlers.Turns;

public class AI {

// length and counter of how long it seems that the player is thinking

private static int \_AIThink;

private static Long \_AIThinkTime;

private static boolean \_finishedTurn;// did the AI finished their act

private static boolean \_tryedToPutBarrier;// did the AI tryed to put barrier but the position was ileagal

private static int \_lastBestRow[] = new int[5], \_lastBestCol[] = new int[5];// last best row and col that were found

// in the algorithm

private static Board \_board;// the game board

private static void decideNextAction() {

// the function purpose is to make the AI play

// in a way that he choose the best act he can do in the current state

Player p = Turns.getCurrTurnPlayer();

\_finishedTurn = p.isFinishedPlay();

// choose and check the state only if the player isnt moving and havnt finished

// the act

if (!(p.is\_moving() || \_finishedTurn)) {

int curturn = Turns.\_currentTurn;

int arr[] = new int[Turns.numTurns];// array of all players shortest path

int min = 0;// the index of the player with min path

// get the len and dir of each player's shortest path to their goal

// and save the index of the player with the shortest path between all players

// in min

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

arr[i] = getPlayerShortestPathLen(Players.\_players[i], i);

if (arr[i] < arr[min])

min = i;

else if (arr[i] == arr[min] && (new Random()).nextBoolean()) {

min = i;

}

}

if (!blockShortestPathPlayer(min, curturn, arr))// if the AI didnt block/couldnt block then make a move

// to his shortest path

p.setDir(Directions.fromPointDir(p.get\_row(), p.get\_col(), \_lastBestRow[curturn],

\_lastBestCol[curturn]));

}

}

private static boolean blockShortestPathPlayer(int min, int curturn, int arr[]) {

// input- the index of the player with the min path, the player who's current

// turn and an array of the shortest path of each players according to the index

// output- flag whether the current player found a best spot to block and

// Proceed to block it

boolean put = false;

Random rnd = new Random();

int ranks[] = new int[Turns.numTurns];

int enemysSum = 0;

int mySum = 0;

// sum all the players state rank

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

ranks[i] = shouldPutBarrier(i, arr);

// set AI rank

mySum = ranks[curturn];

// set enemys rank sum

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

enemysSum += (i == curturn) ? 0 : ranks[i];

// if the AI Rank is that high then he must take a move

boolean should = mySum \* Turns.numTurns - enemysSum \* 2 < 0;

// decide whether the player should try to block or not

if (arr[curturn] > 1 && !\_tryedToPutBarrier && (min != curturn) && (should || rnd.nextBoolean())) {

// try to block the way

put = bestSpotToPutBarrier(min, curturn, arr);

// if the AI cant block the way for the player with the shortest path he will

// search for the next player with the shortest path (who isnt them)

if (!put) {

int num = curturn;

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

if (i != curturn && i != min && arr[i] < arr[num])

num = i;

if (num != curturn)

put = bestSpotToPutBarrier(num, curturn, arr);

}

}

return put;

}

private static boolean bestSpotToPutBarrier(int min, int curturn, int[] arr) {

// input- the index of the player with the min path, the player who's current

// turn and an array of the shortest path of each players according to the index

// output- flag whether the current player found a best spot to block and

// procced to block it

double risk = 1.5;// how far the player will let the blocking spot impact hit path

int row = -1, col = -1;

boolean rotate = false;

int shortestPath = arr[min];

int myPath = arr[curturn];

// horizontal

for (int i = 1; i <= Board.\_rows - 1; i++) {

for (int j = 0; j <= Board.\_cols - 2; j++) {

Barrier b = new Barrier(i, j, \_board, 1, false);

if (\_board.addBarrier(b)) {

LegalChecks.\_Graph.initBarrier(b);

if (LegalChecks.barrierPositionLegal()) {

int temp = getPlayerShortestPathLen(Players.\_players[min], 4);

int mytemp = getPlayerShortestPathLen(Players.\_players[curturn], 4);

// does the barrier delay the other player

if (temp > shortestPath) {

// the barrier shouldnt delay the player who put it if it make a big impact

if (myPath < arr[curturn] \* risk) {

myPath = mytemp;

shortestPath = temp;

row = i;

col = j;

}

}

}

// remove the barrier that used for the checks

\_board.removeBarrier(b);

LegalChecks.\_Graph.initBarrier(b);

}

}

}

// vertical

for (int i = 0; i <= Board.\_rows - 2; i++) {

for (int j = 1; j <= Board.\_cols; j++) {

Barrier b = new Barrier(i, j, \_board, 1, true);

if (\_board.addBarrier(b)) {

LegalChecks.\_Graph.initBarrier(b);

if (LegalChecks.barrierPositionLegal()) {

int temp = getPlayerShortestPathLen(Players.\_players[min], 4);

int mytemp = getPlayerShortestPathLen(Players.\_players[curturn], 4);

// does the barrier delay the other player

if (temp > shortestPath) {

// the barrier shouldnt delay the player who put it if it make a big impact

if (myPath < arr[curturn] \* risk) {

myPath = mytemp;

shortestPath = temp;

row = i;

col = j;

rotate = true;

}

}

}

// remove the barrier that used for the checks

\_board.removeBarrier(b);

LegalChecks.\_Graph.initBarrier(b);

}

}

}

boolean res = false;

if (row != -1 && col != -1)// if a best position was

res = Players.\_players[curturn].putBarrierIn(row, col, rotate);

return res;

}

private static int shouldPutBarrier(int playerNum, int[] arr) {

// input the player num and array of len

// output- the state rank of the player

int decition = 8100;// 81 is the max steps player might need to take to reach the end (its a very

// rare and unlikely to ever occur)

int myBarrierWidth = 3;

int enemyBarrierWidth = 1;

// devide the decition by the length to go

// (cant devide by zero and if it reach zero that mean that the player had won)

int devide = arr[Turns.\_currentTurn] != 0 ? arr[Turns.\_currentTurn] : 1;

// the shorter the len to the end the higher decision to move

decition /= devide;

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

// the more barriers the player have left the more options to block he will have

// later

if (i == playerNum)

decition += Players.\_players[playerNum].get\_barriersLeft() \* myBarrierWidth;

// the more barriers they have the more the player should try to avoid getting

// in a bed situation

// so he might consider to block them instead of move

else

decition -= Players.\_players[i].get\_barriersLeft() \* enemyBarrierWidth;

}

return decition;

}

private static int getPlayerShortestPathLen(Player p, int num) {

// input- a player and his index

// output- his shortest path to his goal and the row and col he better move to

ArrayList<Integer> dirs = new ArrayList<Integer>();

Graph g = LegalChecks.\_Graph;

ArrayList<Edge> e = g.getGraph()[p.get\_row()][p.get\_col()].getAdjacencies();

// check all of the player's possible neighbors and use BFS from there to reach

// their goal

for (int i = 0; i < e.size(); i++) {

int row = e.get(i).getTarget().getI(), col = e.get(i).getTarget().getJ();

dirs.add(BFS.bFS(row, col, p.getGoal(), g));

}

// get the min len between all the neighbors

int min = 0;

for (int i = 0; i < dirs.size(); i++) {

if (dirs.get(min) == -1 || dirs.get(i) != -1 && dirs.get(i) < dirs.get(min))

if (e.get(i).getTarget().getI() >= 0 && e.get(i).getTarget().getJ() >= 0

&& e.get(i).getTarget().getI() < Board.\_rows && e.get(i).getTarget().getJ() < Board.\_cols)

min = i;

}

// set the neighbor with the shortest path row and col

\_lastBestRow[num] = e.get(min).getTarget().getI();

\_lastBestCol[num] = e.get(min).getTarget().getJ();

// return the len of the shortest path

return dirs.get(min);

}

public static void startThink() {

// init the AI turn

init();

startThinking();

}

public static void update() {

// doing the AI act after he had finish "thinking" and didnt finish an act

if (!\_finishedTurn && canDoNextAction())

decideNextAction();

}

public static boolean isFinishedTurn() {

return \_finishedTurn;

}

// timer that make it seen that the ai is thinking

private static boolean canDoNextAction() {

long elapsed = (System.nanoTime() - \_AIThinkTime) / 1000000;

return (elapsed > \_AIThink);

}

private static void startThinking() {

\_AIThink = 1000;

\_AIThinkTime = System.nanoTime();

}

public static void init() {

\_finishedTurn = false;

\_tryedToPutBarrier = false;

}

public static void initBoard(Board b) {

\_board = b;

}

public static void tryToPutBarrier() {

\_tryedToPutBarrier = true;

}

}

מחלקת BFS

package Logic;

import java.awt.Point;

import java.util.ArrayList;

public class BFS {

//is there away to reach the goal/ did it reach it

private static boolean \_reachEnd;

//visited cells matrix

private static boolean \_visited[][];

//how many move/ laps the algorithm took

private static int movCount;

//how many nodes/cells are currently in progres to reach the end in the current move/step/lap

private static int nodesLeftInLayer;

//how many nodes we can check in the next lap/step/mov

private static int nodesInNextLayer;

//init all the parameters for a new search, then call the actual BFS algorithm

public static int bFS(int row,int col,Point goal, Graph g) {

movCount = 0;

nodesLeftInLayer = 1;//current point

nodesInNextLayer = 0;

\_reachEnd = false;

\_visited = new boolean[g.get\_rows()][g.get\_cols()];

//init visited matrix cells to false(we havent visited any cell yet)

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

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

\_visited[i][j] = false;

}

}

//call the actual BFS

boolean flag = solve(row, col,goal, g);

//if the algorithm found way to reach the end, return how many steps it took

if (flag)

return movCount;

//if it didnt find a way then return -1

return -1;

}

private static boolean solve(int row, int col,Point goal, Graph g) {

// remove 0 and add last = Queue

ArrayList<Integer> rq = new ArrayList<Integer>();//rows Queue

ArrayList<Integer> cq = new ArrayList<Integer>();//cols Queue

//add current point

rq.add(row);

cq.add(col);

//mark this point as visited

\_visited[row][col] = true;

//while there arent any points left and we havent reach the goal

while (!\_reachEnd&& rq.size() > 0 && cq.size() > 0) {

//check the node in the current layer

row = rq.remove(0);

col = cq.remove(0);

//if it acctualy the goal than exit with true

if (row == goal.x||col==goal.y) {

\_reachEnd=true;

return true;

}

//check what node's neighbours we can add to the next layer

explore\_neighbours(row, col, rq, cq, g);

//moving to the next node in this layer

nodesLeftInLayer--;

//if the layer ended we need to check their childern

if (nodesLeftInLayer == 0) {

//the next layer turning into the current layer

nodesLeftInLayer = nodesInNextLayer;

//init next layer counter

nodesInNextLayer = 0;

//increase progres move

movCount++;

}

}

//if the algorithm reach this point it means it didnt find a way to reach the goal

return \_reachEnd;

}

//a function that adds the nodes the algorithm can visit in the next layer

private static void explore\_neighbours(int row, int col, ArrayList<Integer> rq, ArrayList<Integer> cq, Graph g) {

ArrayList<Edge> e = g.getGraph()[row][col].getAdjacencies();//array of edges/cells that the player can go to from the row col- cell

for (int i = 0; i < e.size(); i++) {

//get the position of the neighbour

int rr = e.get(i).getTarget().getI();

int cc = e.get(i).getTarget().getJ();

//if we visited the node then we shouldnt add it to the next layer again (we already treated this node)

if (\_visited[rr][cc])

continue;

//if we didnt visited the node then add its point

rq.add(rr);

cq.add(cc);

//mark as visited

\_visited[rr][cc] = true;

//the nodes in the next layer have been increased

nodesInNextLayer++;

}

}

}

מחלקת Edge

package Logic;

public class Edge {

private final Vertex \_target;// the end of the edge

private final double \_weight = 1;// the weight

public Edge(Vertex target) {

\_target = target;

}

public Vertex getTarget() {

return \_target;

}

public double getWeight() {

return \_weight;

}

}

מחלקת Graph

package Logic;

import java.util.ArrayList;

import Board.Board;

import Board.Directions;

import Entity.Barrier;

public class Graph {

private Vertex[][] \_matVert;// the Dijkstra map

private int \_rows;// the rows

private int \_cols;// the colums

public static Board \_Board;

public Graph(Board b) {

\_Board = b;

\_rows = b.\_rows;

\_cols = b.\_cols;

\_matVert = new Vertex[\_rows][\_cols];

for (int i = 0; i < \_rows; i++) {

for (int j = 0; j < \_cols; j++) {

\_matVert[i][j] = new Vertex(i, j);

}

}

buildGraph();

}

public int get\_rows() {

return \_rows;

}

public int get\_cols() {

return \_cols;

}

public Vertex[][] getGraph() {

//PrintGraphics();

return \_matVert;

}

public void buildGraph() {

//init all the vertexes and edges in the graph

for (int i = 0; i < \_rows; i++) {

for (int j = 0; j < \_cols; j++) {

\_matVert[i][j].getAdjacencies().clear();

ArrayList<Edge> adjacencies = new ArrayList<Edge>();

for (int d = Directions.NumDirs - 2; d >= 0; d -= 2) {

if (!\_Board.canGoTo(i, j, d))

continue;

int rr = i + Directions.Dirsy[d];

int cc = j + Directions.Dirsx[d];

adjacencies.add(new Edge(\_matVert[rr][cc]));

}

\_matVert[i][j].setAdjacencies(adjacencies);

}

}

}

public void initBarrier(Barrier b) {

//input- a barrier

//outcome- the graph is disconnect/connect all the vertexes and edges in the area of the barrier

int row = b.get\_row();

int col = b.get\_col();

for (int i = -1; i < 2; i++)

for (int j = -1; j < 2; j++) {

int r = row+i;

int c =col+j;

if(r<0 || c<0 ||r>=\_rows ||c>=\_cols)

continue;

\_matVert[r][c].getAdjacencies().clear();

ArrayList<Edge> adjacencies = new ArrayList<Edge>();

for (int d = Directions.NumDirs - 2; d >= 0; d -= 2) {

if (!\_Board.canGoTo(r, c, d))

continue;

int rr = r + Directions.Dirsy[d];

int cc = c + Directions.Dirsx[d];

adjacencies.add(new Edge(\_matVert[rr][cc]));

}

\_matVert[r][c].setAdjacencies(adjacencies);

}

}

}

מחלקת LegalChecks

package Logic;

import Entity.Player;

public class LegalChecks {

private static Player *\_players*[];// players array

public static Graph *\_Graph*;// the game graph

public static void init(Player p[], Graph b) {

// init

*\_players* = p;

*\_Graph* = b;

}

public static boolean barrierPositionLegal() {

// go throh all the players and check if they have a way to reach their goal

// if not return false so the game want let the last barrier position to be

// legal

boolean flag = true;

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

if (BFS.*bFS*(*\_players*[i].get\_row(), *\_players*[i].get\_col(), *\_players*[i].getGoal(), *\_Graph*) == -1) {

flag = false;

}

}

return flag;

}

}

מחלקת Vertex

package Logic;

import java.util.ArrayList;

public class Vertex {

private ArrayList<Edge> \_adjacencies;// the adjacencies

private double \_minDistance = Double.***POSITIVE\_INFINITY***;// the minimum distance

private Vertex \_previous;// the previous vertex

private int \_i, \_j;

public Vertex(int i, int j) {

\_adjacencies = new ArrayList<Edge>();

\_i = i;

\_j = j;

}

public int getI() {

return \_i;

}

public int getJ() {

return \_j;

}

public ArrayList<Edge> getAdjacencies() {

return \_adjacencies;

}

public void setAdjacencies(ArrayList<Edge> adjacencies) {

\_adjacencies = adjacencies;

}

public double getMinDistance() {

return \_minDistance;

}

public void setMinDistance(double minDistance) {

\_minDistance = minDistance;

}

public Vertex getPrevious() {

return \_previous;

}

public void setPrevious(Vertex previous) {

\_previous = previous;

}

public int compareTo(Vertex other) {

return Double.*compare*(\_minDistance, other.getMinDistance());

}

}

מחלקת GamePanel

package Main;

import javax.swing.JPanel;

import java.awt.\*;

import java.awt.image.BufferedImage;

import java.awt.event.\*;

import javax.swing.JPanel;

import Handlers.Keys;

import GameState.GameStateManager;

@SuppressWarnings("serial")

public class GamePanel extends JPanel implements Runnable, KeyListener {

// dimensions

public static int WIDTH = 1;

public static int HEIGHT = 1;

public static final int SCALE = 2;//make images bigger

// game thread

private Thread \_thread;

private boolean \_running;

private int \_FPS = 60;

private long \_targetTime = 1000 / \_FPS;

// image

private BufferedImage \_image;

private Graphics2D \_g;

// game state manager

private GameStateManager \_gsm;

public GamePanel(int w, int h) {

super();

WIDTH = w / SCALE;

HEIGHT = h / SCALE;

setPreferredSize(new Dimension(WIDTH \* SCALE, HEIGHT \* SCALE));

setFocusable(true);

requestFocus();

}

public void addNotify() {

// init threat and listeners

super.addNotify();

if (\_thread == null) {

\_thread = new Thread(this);

addKeyListener(this);

addMouseListener(new MouseAdapter() {

@Override

public void mouseClicked(MouseEvent e) {

super.mouseClicked(e);

MouseClicked(e);

}

});

\_thread.start();

}

}

private void init() {

// in my game I have a global Graphics variable that I use it to draw the pages

// on the screen

// the use of it help to define which page is the one that should draw

// it helps and more comfortable for me to work like it

// so every frame the current page is drawn to the graphics of the global g

// and then the g graphics is drawn to the screen

// it also can benefit when changing the resolution

// or to choose different color approach

// which in my case its a basic RGB

\_image = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE\_INT\_RGB);

\_g = (Graphics2D) \_image.getGraphics();

\_running = true;

\_gsm = new GameStateManager();

}

public void run() {

init();

long start;

long elapsed;

long wait;

// game loop

while (\_running) {

start = System.nanoTime();

// the threat should always update the game draw the page to the global g and

// then to the screen

update();

draw();

drawToScreen();

elapsed = System.nanoTime() - start;

wait = \_targetTime - elapsed / 1000000;

if (wait < 0)

wait = 5;

try {

Thread.sleep(wait);

} catch (Exception e) {

e.printStackTrace();

}

}

}

private void update() {

// update the page/state manager and the keys

\_gsm.update();

Keys.update();

}

private void draw() {

// draw to the global g

\_gsm.draw(\_g);

}

private void drawToScreen() {

// draw to the screen in the size of the screen

Graphics g2 = getGraphics();

g2.drawImage(\_image, 0, 0, WIDTH \* SCALE, HEIGHT \* SCALE, null);

g2.dispose();

}

public void keyTyped(KeyEvent key) {

}

public void keyPressed(KeyEvent key) {

//set key pressed

Keys.keySet(key.getKeyCode(), true);

}

public void keyReleased(KeyEvent key) {

//set key released

Keys.keySet(key.getKeyCode(), false);

}

public void MouseClicked(MouseEvent e) {

//set mouse clicked

\_gsm.MouseClicked(e);

}

}

מחלקת Main

package Main;

import java.awt.Dimension;

import java.awt.Toolkit;

import javax.swing.JFrame;

public class Main {

public static void main(String[] args) {

// screen size

Dimension DimMax = Toolkit.getDefaultToolkit().getScreenSize();

// jframe settings

JFrame window = new JFrame("Quoridor");

window.setUndecorated(true);

window.setResizable(false);

// set full screen

window.setMinimumSize(DimMax);

window.setExtendedState(JFrame.MAXIMIZED\_BOTH);

window.setLocationRelativeTo(null);

window.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

window.add(new GamePanel(DimMax.width, DimMax.height));

window.pack();

window.setVisible(true);

}

}

## **18. רפלקציה**

לאחר גמר הפרוייקט שלי אני מרגיש כאילו למדתי יותר להסתמך על עצמי וללמוד איך לחשוב מחוץ לקופסא העבודה על הפרוייקט היית מאוד מהנה. חוץ מהחומר הנלמד בכיתה לא השתמשתי בעוד הרבה מקורות למידה וניסיתי להיעזר כמה שיותר רק בכוחות עצמי ולהתייעץ עם אחרים על דברים בסיסים כמו עיצוב המשחק, על מנת לאתגר את עצמי ולראות לאיזה יכולות אני מסוגל להגיע עם בכוחות עצמי ביצירת פרוייקט. כתוצאה מכך אני מרגיש שאופי הפרויקט שלי והתנהלותו קצת יוצא דופן מפרויקטים אחרים.

בזמן כתיבת הפרוייקט נעמדתי בכמה קשיים. על רובם התגברתי והסתדרתי איתם אך הקושי העיקרי בעייני שעדיין יכול לבלבל את המשתמש הוא צורת קבלת INPUT מהמשתמש, אני מודע זהוי הפעם הראשונה(וחצי- בנוסף לפרוייקט שהתחלנו ליצור בתחילת שנה) שיצא לי לשלב מערכת של האזנה למקשי המקלדת והעכבר יחד. מה שאולי יכול לבלבל את המשתמש באיזה שלב הוא צריך\יכול להשתמש במה, אך השתדלתי שהפרוייקט יהיה מהנה ונוח ככל האפשר.

התקשתי גם למצוא תמונות\sprites או דומויות וכפתורים שמצאו כן בעייני לאופי של הפרוייקט שלי, לכן חלק מהכפתורים , הודעות הנצחון והמסך הראשי הם היו בעיצוב שלי.  
למרות שאיני שבע רצון מהכל אני חושב שלבסוף זה השתלב בצורה שנראת אסתטית לעיין.

אם הייתי צריך לעשות את הפרוייקט מההתחלה, אני חושב שהייתי מנסה להתחיל את העבודה על הפרוייקט בצורה יותר מסודרת ואולי גם יותר יעילה למרות שאני מאוד אוהב את איך שרוב הפעולות והקוד שלי יצא אני חשוב שעוד קצת סדר בזה לא יזיק.

כמובן שהייתי רוצה שהמחשק יהיה יותר אסתטי ומסודר אך בכל זאת אני עדיין מרוצה מהדרך שהוא יצא והתפתח.