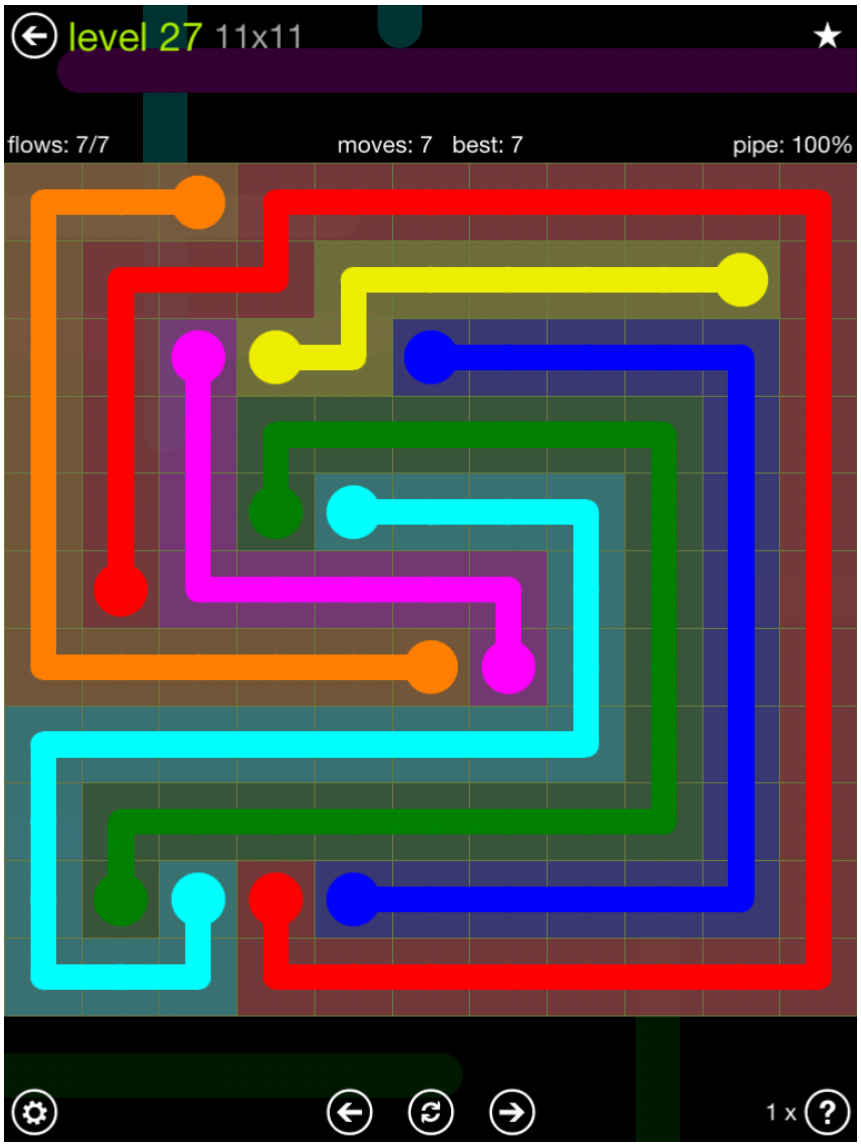
פרוייקט קורס מבוא לבינה מלאכותית
FLOW FREE

ואיל שאמא כריסתיאן מטאנס



תוכן עניינים

הגדרת המשחק

במשחק הזה יש לוח משובץ עם זוגות של נקודות בצבעים שונים. המטרה היא למצוא מסלולים שיקשרו בין כל שתי נקודות בעלות אותו הצבע, מבלי שהמסלולים יחתכו אחד את השני. דרישה נוספת היא שהמסלולים יכסו את כל הלוח כך שלא ישארו בו משבצות ריקות, הבעיה זהה לפאזל שנקרא number-link שהוכח עבור שהיא בעיה היא NP שלמה. (מקור : https://www.jstage.jst.go.jp/article/ipsjjip/23/3/23_239/_article)

המשחק עובד על כישורים מרחביים-חזותיים, כמו גם פתרון בעיות, תכנון שלבי העבודה ועוד. אחד הדברים הנחמדים שאפשר לעבוד עליהם דרכו הוא גמישות מחשבה. יש שחקנים שנתקעים בשלבים מסוימים כשהם מנסים שוב ושוב ליצור מסלולים באותו סגנון עבודה או סדר פעולות שבו השתמשו בהצלחה בשלבים קודמים. לכאורה יש יתרון לתת קדימות לאסטרטגיה מנצחת, אבל לפעמים משתנות הנסיבות וכדי להתקדם יש להכיר בצורך לשינוי גישה. מצב זה מזמן תרגול בשבירת מוסכמות על ידי שימוש באסטרטגיה רנדומלית אך מחושבת היטב.
על פי רוב, ככל ששחקן משתמש בה יותר כך הוא גם מצמצם את הצורך בה. זאת משום שהרחבת אפשרויות הפתרון העומדות לרשותו מצמצמות את כמות השלבים שבהם הוא נתקע.

CSP – Constraint Satisfaction Problem

Constraint satisfaction problems (CSPs) are mathematical problems defined as a set of objects whose state must satisfy a number of constraints or limitations.

● CSPs represent the entities in a problem as a homogeneous collection of finite constraints over variables, which is solved by constraint satisfaction methods.

● CSPs are the subject of intense research in both artificial intelligence and operations research, since the regularity in their formulation provides a common basis to analyze and solve problems of many seemingly unrelated families.

● CSPs often exhibit high complexity, requiring a combination of heuristics and combinatorial search methods to be solved in a reasonable time.

Formal Constraint Satisfaction Problem

Formal Constraint Satisfaction Problem

Given:

- A set of variables $\mathcal{X} = \{x_i\}_1^N$
- A set of domains $\mathcal{D} = \{D_i\}_1^N$
- A set of constraints $\mathcal{C} = \{C_j\}_1^K$ $where$ $C_j \subseteq D_1 \times ... \times D_N$

Find: a substitution $\{x_i \leftarrow d_i\}$, s.t. :

- $\forall\, 1 \leq i \leq N$, $d_i \in D_i$
- $\forall\, 1 \leq i \leq K$, $(d_1, ..., d_N) \in C_i$

Note that constraints can be given in a functional form:
$C_j : D_1 \times ... \times D_N \rightarrow \{0, 1\}$

- $\forall\, 1 \leq i \leq K$, $C_j(d_1, ..., d_N) = 1$

הגדרת הבעיה שלנו כ-CSP

בבעייה שלנו הגדרנו את **המשתנים(variables)** להיות המשבצות בתוך הלוח.
חילקנו את סוגי המשתנים לשלושה קטיגוריות:

1. משתנים מסוג מקור שהם מייצגים את המשבצות הראשונות בלוח ההתחלתי מצבע מסויים (אם הולכים על הלוח משמאל לימין, מלמעלה למטה).
2. משתנים מסוג בור שהם מייצגים את המשבצות האחרונות בלוח ההתחלתי מצבע מסויים (אם הולכים על הלוח משמאל לימין, מלמעלה למטה).
3. משתנים האחרים מייצגים את המשבצות הריקות בלוח ההתחלתי.

את **התחומים (domains)** של כל משתנה הגדרנו בהתאם לסוגו:

- אם הוא משתנה רגיל (לא בור ולא מקור), קבענו שנציב בו את הוקטור הבא :
(שכן קודם, צבע, שכן הבא) . לכן התחום של משתנה מסוג זה הוא מכפלה קרטזית של הקבוצות הבאות, {קבוצת השכנים של המשתנה}X{קבוצת הצבעים}X{קבוצת השכנים של המשתנה}, כך שהשכן הקודם לא שווה לשכן הבא. בסך הכל גודל התחום של משתנה מסוג זה הוא N*3 = 12*N*4 . (כאשר N הוא מספר הצבעים)

- אם הוא מקור אז המשבצת כבר צבועה ולא יכול להיות לו 'שכן קודם' במסלול אז ההצבה בו היא רק עבור מה יהיה השכן הבא שלו ולכן תחום הערכים שלו הוא המשבצות השכנות לו והתחום מגודל מקסימלי של 4 .

- אם הוא בור אז המשבצת כבר צבועה ולא יכול להיות לו 'שכן הבא' במסלול אז ההצבה בו היא רק עבור מה יהיה השכן הקודם שלו ולכן תחום הערכים שלו הוא המשבצות השכנות לו והתחום מגודל מקסימלי של 4 .

*הגדרת שכן : שכן של משבצת הוא משבצת אחרת בתוך הלוח שנמצאת מתחתיה, מעליה מימינה או משמאלה.

את **האילוצים**(**constraints**) של כל משתנה הגדרנו גם בהתאם לסוג המשתנה:

- אם מדובר במשתנה רגיל, האילוצים הם על ערכי השכן הקודם ,הצבע, והשכן הבא.

האילוץ על השכן הקודם:

1. אסור שהשכן הקודם יהיה משבצת שהיא בור.
2. אם מוצב בשכן הקודם צבע, אסור שהוא יהיה שונה מהצבע המוצע בהשמה עבור משתנה זה.
3. אם הצבע של השכן הקודם שווה לצבע המוצע, ומוצב בשכן הקודם ערך השכן הבא שלו, אסור שהשכן הבא של השכן הקודם יהיה שונה מהמשבצת הנוכחית עבורה אנו מנסים למצוא הצבה.
4. אם המשבצת הנוכחית היא השכן הבא של אחד מהשכנים ,הערך של השכן הקודם המוצע עבור המשבצת הוכחית אסור שיהיה שונה מאותו שכן .

האילוץ על השכן הבא:

1. אסור שהשכן הבא יהיה משבצת שהיא מקור.
2. אם מוצב בשכן הבא צבע, אסור שהוא יהיה שונה מהצבע המוצע בהשמה עבור משתנה זה.

3. אם הצבע של השכן הבא שווה לצבע המוצע, ומוצב בשכן הבא ערך השכן הקודם שלו, אסור שהשכן הקודם של השכן הבא יהיה שונה מהמשבצת הנוכחית עבורה אנו מנסים למצוא הצבה.
4. אם המשבצת הנוכחית היא השכן הקודם של אחד מהשכנים ,הערך של השכן הבא המוצע עבור המשבצת הוכחית אסור שיהיה שונה מאותו שכן .

<u>אילוץ על משתנה שהוא בור</u> דומה לאילוץ על השכן הקודם עבור משתנה רגיל, בתוספת שאסור שהמשתנה הקודם יהיה מקור(עבור בור מציבים רק מה הוא השכן הקודם).

<u>האילוץ על משתנה שהוא מקור</u> דומה לאילוץ על השכן הבא עבור משתנה רגיל, בתוספת שאסור שהמשתנה הבא יהיה בור(עבור מקור מציבים רק מה הוא השכן הבא)
הבהרות נוספות עבור האילוצים ניתן למצוא בהסבר על המימוש בהמשך.

המימוש

חשבנו לפתור את בעיית FLOW FREE דרך CSP שלמדנו בכיתה כך שמצאנו שהבעיה היא בעיה מתימטית שאפשר לבטא אותה באמצעות משתנים , תחומי משתנים , ואילוצים. וגם הצלחנו להגדיר (heuristic methods) שיטות ופונקציות עזר, כך שנוכל להפחית את זמן הריצה של פתירת לוחות, ספיציפית לוחות עם סיבוכיות גבוהה יחסית.

הצלחנו להגדיר שיטות עזר כגון לבחור כל פעם את המשתנה שיש לו הכי פחות ערכים(Minimum Remaining Value) ו את השיטות (Degree Heuristic) ו (Least Constrained Value) ו (Forward Checking).

ויכלנו להוסיף עוד שיטות עזר שחשבנו עליהם לפני שהתחלנו לממש את קוד ה CSP וגם המצאנו כמה תוך כדי הניסוי שלנו בפתירה , כגון סדר עבירה על האובייקטים מסוג VARIABLE , ושיטות להפחית את הדומיין של משתנה בשלבים מסויימים , ושיטות מפחיתות לנו במקרים רבים את זמן הריצה תוך כדי ה-BackTracking.

מחלקת VARIABLE

VARIABLE הוא אובייקט שמייצג את המרובעים בלוחות (צבועים וריקים) שהם המשתנים בהגדרת הבעיה.
כל מרובע הוא משתנה בבעייה , כך שהאובייקט מסוג VARIABLE מכיל את :

- מקומו בלוח (i,j) - tuple
- צבע המרובע
- תחום המשתנה (DOMAIN) – הערכים האפשריים.
- ערכים החוקיים לכל משתנה (legal Values)
- IsSource Flag – אם הוא מקור* של צבע מסויים.
- IsTarget Flag – אם הוא בור* של צבע מסויים.
- מערך של השכנים של המשתנה (neighbors) .
- ה-NEXT** של המשתנה , הוא גם מאובייקט VARIABLE.
- ה-PREVIOUS** של המשתנה , הוא גם מאובייקט VARIABLE.

*** לכל צבע בלוח הראשוני יש מקור ו בור כך שמחוקיות המשחק צריך להעביר מסלול בין כל מקור ובור.**

**** הגדרנו את הבעיה כ - "linked List" כך שלכל משתנה בסדרה יש next ו previous כך שנשמור על מסלולים בין כל מקור ובור, חוץ מהמקור והבור , שלמקור יש רק next והבור יש לו רק previous.**

<u>הערה</u>
כל מרובע הוא מייצג VARIABLE שונה , כך שכל מרובע צריך למלא לו בסוף הפתרון את הצבע שלו ואת ה previous וה next כך שלא יהיה אף מרובע שלא שייך לאף מסלול.וכך נשמור על האילוץ השני של חוקיות הפתרון "המסלולים יכסו את כל הלוח כך שלא ישארו בו משבצות ריקות".

Constraints

האילוצים מחולקים ל-3 סוגים של משתנים, והם יהיו אילוצים על ההשמה הנוכחית של המשתנה.
האילוצים על ההשמות הלא נכונות , כלומר מחזירים FALSE רק כאשר לא מתקיימים תנאים מסויימים, אחרת TRUE.

1) אובייקטים שמקיימים IsSource :
: המשתנים הללו יש להם צבע נתון מהגדרת הלוח , ולכן האילוצים הם

- אם קיים אחד מהשכנים שלו שה- previous שלו הוא אותו משתנה ובהשמה הנוכחית לא מגדירים את ה next לשכן הזה מחזירים FALSE.
- אם בהשמה הנוכחית מצביעה על מקור או בור אחר נחזיר FALSE.
- אם בהשמה הנוכחית מצביעה על משתנה שהושם, אז אסור שנצביע עליו אם הקודם לו משתנה אחר.
- ההשמה צריכה להיות אותו צבע שהוא הוגדר בו מההתחלה.

2) אובייקטים שמקיימים IsTarget :
: המשתנים הללו יש להם צבע נתון מהגדרת הלוח , ולכן האילוצים הם

- אם קיים אחד מהשכנים שלו שה- next שלו הוא אותו משתנה ,ובהשמה הנוכחית לא מגדירים את ה previous לשכן הזה מחזירים FALSE.
- אם בהשמה הנוכחית מצביעה על מקור או בור אחר נחזיר FALSE.
- אם בהשמה הנוכחית מצביעה על משתנה שהושם, אז אסור שנצביע עליו אם העוקב לו משתנה אחר.
- ההשמה צריכה להיות אותו צבע שהוא הוגדר בו מההתחלה.

3) אובייקטים שלא מקיימים IsTarget או IsSource (כלומר משבצות ריקות הוגדרו מהתחלה) – למשתנים הללו יהיה עוקב וקודם ולכן ישלבו בין אילוצי הבור והמקור :

- אם קיים אחד מהשכנים שלו שה- next שלו הוא אותו משתנה ,ובהשמה הנוכחית לא מגדירים את ה previous לשכן הזה מחזירים FALSE.
- אם בהשמה הנוכחית מצביעה previous על מקור נחזיר FALSE.
- אם בהשמה הנוכחית מצביעה על משתנה שהושם, אז אסור שנצביע עליו אם העוקב לו משתנה אחר.
- אם קיים אחד מהשכנים שלו שה- previous שלו הוא אותו משתנה ובהשמה הנוכחית לא מגדירים את ה next לשכן הזה מחזירים FALSE.
- אם בהשמה הנוכחית מצביעה next על מקור נחזיר FALSE.
- אם בהשמה הנוכחית מצביעה על משתנה שהושם, אז אסור שנצביע עליו אם הקודם לו משתנה אחר.

האלגוריתם לפתירת הבעיה (למדנו בכיתה)

Backtracking Search

Function: Backtracking Search
Input: A CSP problem X, D, C
Output: Either a solution or a failure

```
i ← 1                                   // initialize variable counter
D'_i ← D_i                              // copy domain
while 1 ≤ i ≤ n do
    x_i ← SELECT-VALUE                  // no value was returned
    if x_i is null then
        i ← i − 1                       // backtrack
    else
        i ← i + 1                       // step forward
        D'_i ← D_i
    end if
end while
if i = 0 then return "failure"
else
    return x_1, ..., x_n
end if
```

Backtracking Search

Function: SELECT-VALUE
Output: A value in D'_i consistent with $\vec{a}_{i-1}$

```
while D'_i is not empty do
    select an arbitrary element a ∈ D'_i and remove a from D'_i
    if a is consistent with a⃗_{i−1} then return a
    end if
end while
return null
```

Used Heuristics

השתמשנו בכל מיני היוריסטיקות כדי לשפר את הביצוע, ולהפחית את זמן הביצוע למינימלי ככל האפשר כפי שהגדרנו את הבעיה.

1. Minimum Remaining Values(variable ordering heuristic)
קודם כל ממיינים את המשתנים לפי משתנים עם הדומיין הקטן ביותר(הכי פחות ערכים חוקיים), ואחר כך נותנים השמה למשתנה הראשון , ואחר כך בכל שלב בוחרים במשתנה עם הדומיין הקטן ביותר.

2. Minimum Reamaining Values + Heuristic
פה הוספנו עוד היוריסטיקה שהיא variable ordering והיא תמיין את המשתנים בנוסף ל MRV , לפי מספר השכנים הלא מוצבים של המשתנה , דבר זה יעזור לנו לפרוש פחות את החיפוש באמצעות backtracking.

3. Minimum Reamaining Values + Degree Heuristic + Heuristic
משתמשים בהירוסטיקה שנזכרה לעיל , ובמקרה שיש יותר ממשתנה אחד עם דומיין הכי קטן שווה , אז בוחרים במשתנה שיש לו הכי הרבה שכנים שעדיין לא הושמו (כך שיהיה המשתנה עם הכי הרבה אילוצים על אחרים).

עד כאן היוריסטיקות שהן variable ordering.

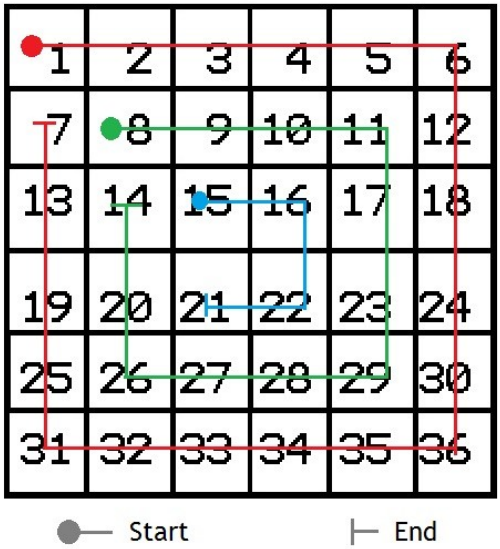
4. Least Constrained Value
היוריסטיקה הזאת היא מסוג value ordering, היא מחזירה את הערך שמאלץ כמה שפחות את המשתנים האחרים, כדי לבדוק זאת במקרה שלנו הנחנו הצבה של המשתנה עם ערך מסויים, ואזעבור שכניו סכמנו את הגדלים של ה domains שהם קונסיסטנטיים עם ההצבה של ערך זה , החזרנו את הערך עם הסכום הגדול ביותר, כי אחרי הצבתו שכניו ישארו עם דומיין קונסיסטינטי גדול יותר, כלומר ההצבה הזאת מגבילה אותם הכי פחות.

5. Forward Checking
look-ahead heuristic, אחרי כל הצבה אנו מאתחלים את השכנים ל domain קונסיסטינטי עם ההצבה , אם ה domain יהיה ריק לא נשים את ההצבה הזאת ונעשה backtracking.

Not Used Heuristics

1) Spiral Heuristic

ממיינים את המשתנים כך שנעבור בקצות קודם כל ונכנס למשתנה שנמצא בדיוק באמצע הלוח. כאשר אנו עוברים צמוד לקירות, זה מפחית מסלולים נחתכים, כי פחות עוברים בצבעים אחרים. וגם כן אנו עוברים על המשתנים כך שכל אחד באיטרציה הוא סמוך לקודם לו ולעוקב לו ,
וזה נותן עדיפות כי כאשר אנו נותנים השמה למשתנה מסויים אז נותנים לו צבע ועוקב והשמה, אז הדומיין של הסמוך לו בלוח יהיה קטן יותר ובסיכוי גדול הוא יימצא התקלות מוקדם יחסית.
אחרי שמימשנו את אופן סידור המשתנים לפי מה שנזכר, ראינו שזה לא ממש עוזר ואם זה יעזור זה רק במקרים ספיציפיים ואופן הסידור דיי דומה ל backTracking שעובר שורה שורה(ממשתנה לשכן שלו).

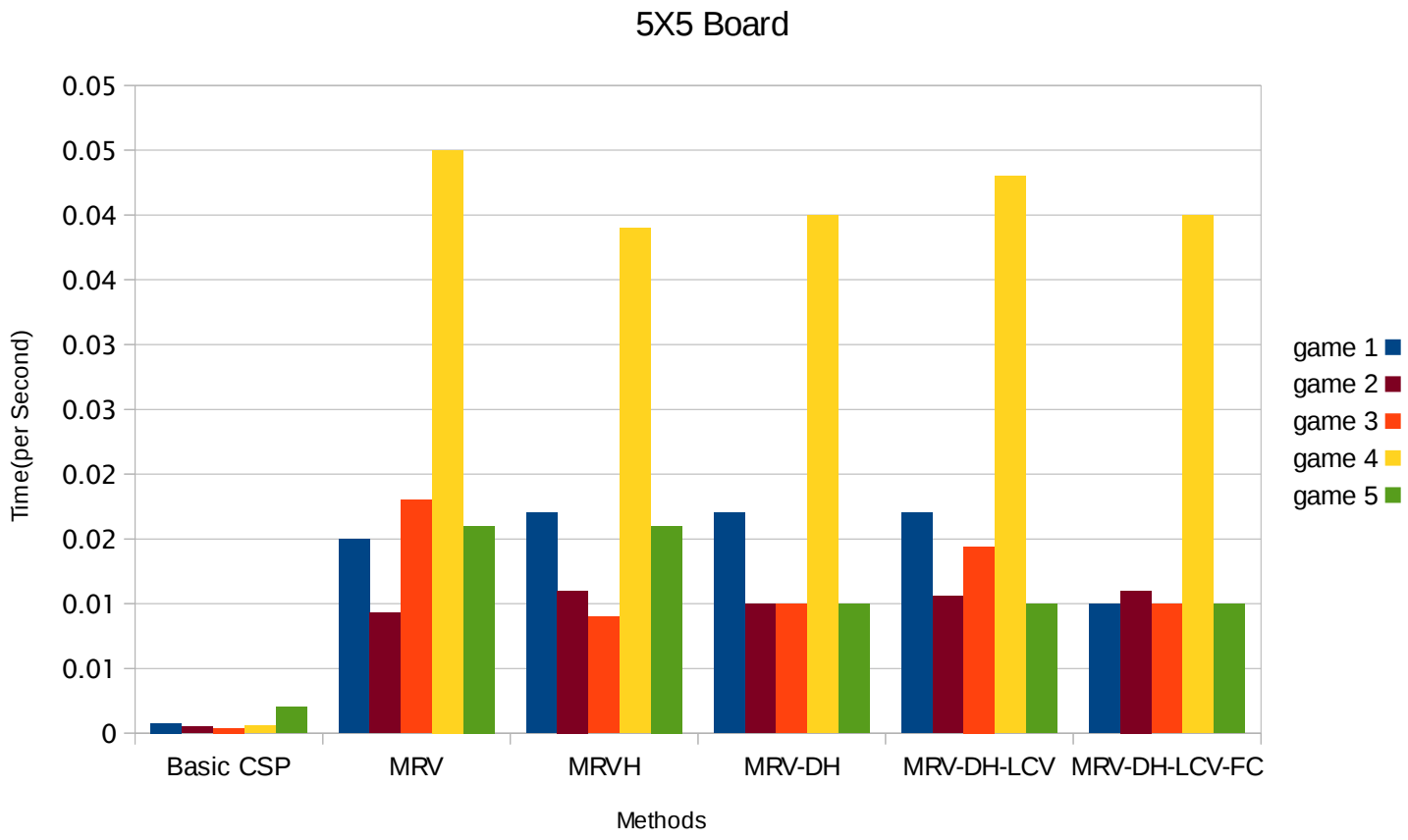


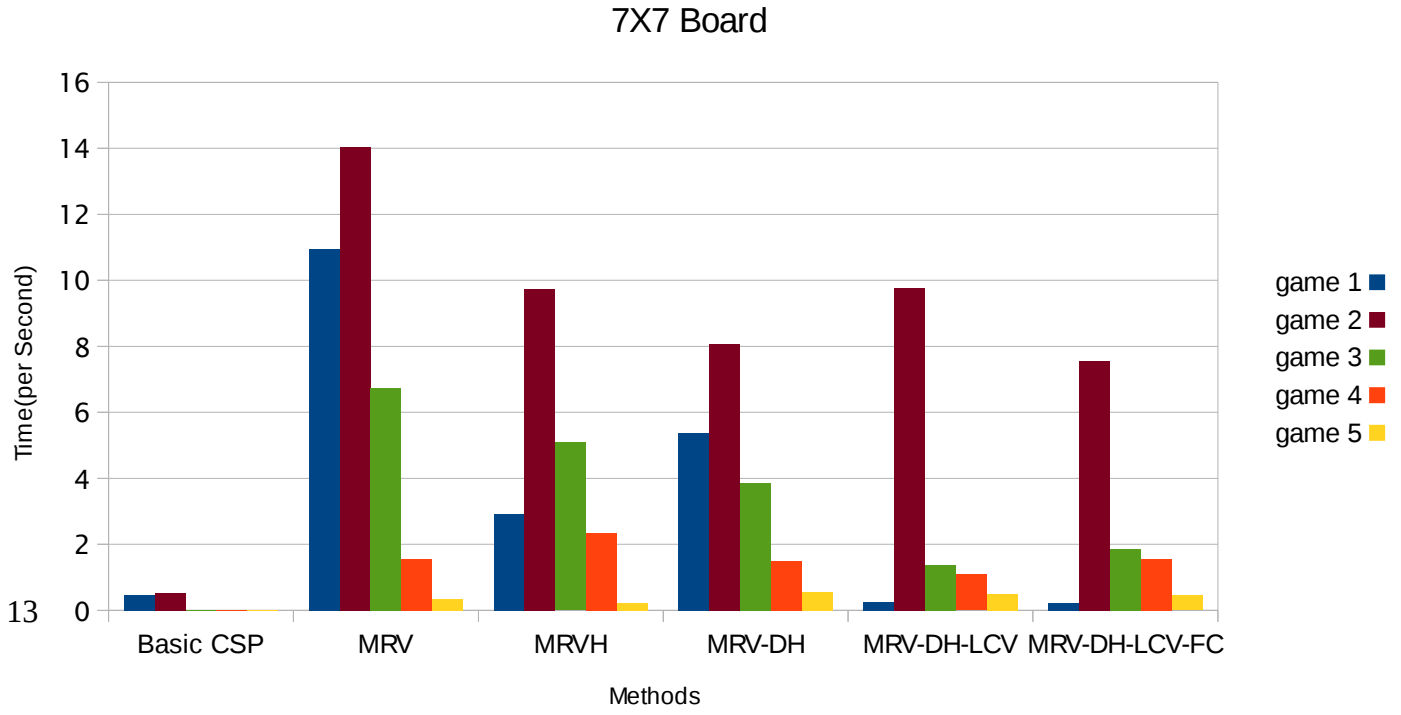
2) Is-Path

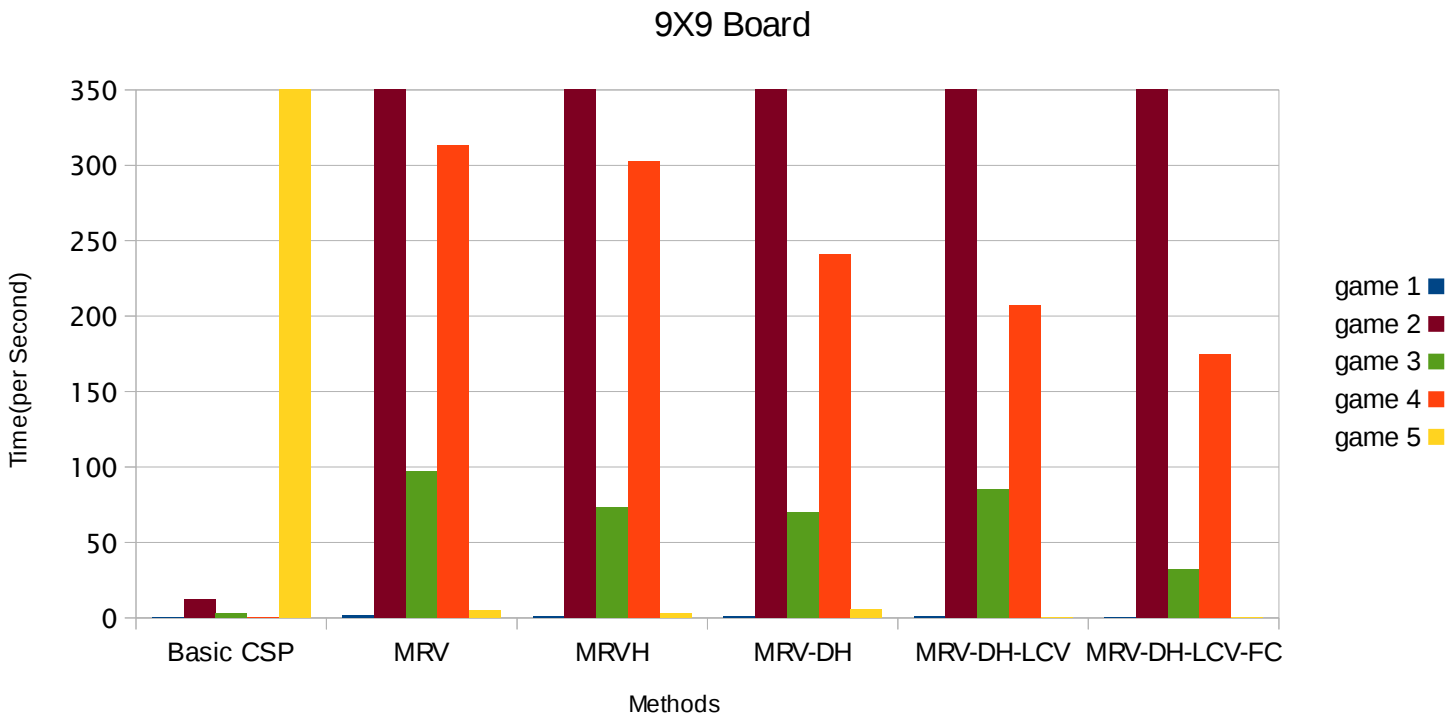
ההיוריסטיקה בודקת בכל שלב באיטרציה אם יש מסלול נוכחי בין כלשהו מקור ובור לצבע מסויים, אם כן אז לא תאפשר למשתנה אחר שניישם מאוחר יותר להיות בצבע ההוא, וכך מפחיתים הדומיין לכל משתנה שיהיה צבע שונה לכל צבע שיש לו כבר מסלול בין מקורו לבורו.
אבל ראינו שבדיקות כללו לוקחות זמן רב , וברוב המקרים המקור נמצא בשורות הראשונות והבור נמצא בשורות האחרונות, אזי עד שיהיה מסלול ביניהם, יישאר לנו מספר משבצות לא מיושמות קטן , ולכן במקרים הללו אנחנו בדקנו לכל צבע אם יש מסלול וגם לכל המשבצות שעברנו עד כה וזה לקח הרבה זמן ולא ייעל בסוף את זמן הפתירה.

תוצאות

ייצרנו 5 משחקים לכל אחד מהשלבים 5X5, 7X7, 9X9. וייצרנו קבצים להרצת כל היוריסטיקה על כל שלב.







מפתח לגרף

Basic-CSP = csp problem with backtracking
MRV = Minimum Remaining Values
MRVH = Minimum Remaining Values with Heuristic
MRV-DH = Minimum Remaining Values, Degree Heuristic when Tie,with Heuristic
MRV-DH-LCV = Minimum Remaining Values, Degree Heuristic when Tie,with Heuristic.And the values are orderd by Least Constrained Values.
MRV-DH-LCV-FC = Minimum Remaining Values, Degree Heuristic when Tie,with Heuristic.And the values are orderd by Least Constrained Values. Forward Checking while Backtracking and moving forward.

הערה לגבי התוצאות בגרף 9X9

בלוח 9X9 היו כמה לוחות שלא נפתרו בזמן סביר, אזי בגרף הזמן שלהם מסומן כ 350

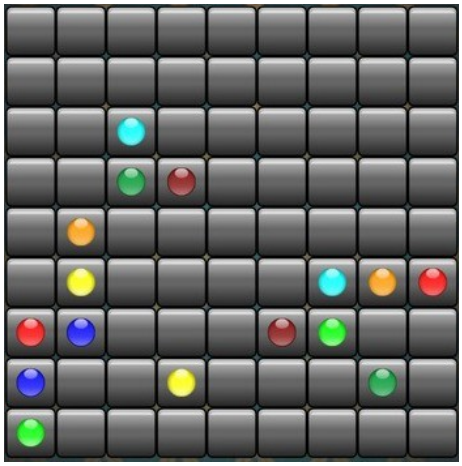
שניות, והוא אפילו יותר מזה .

אנליזה לתוצאות

•אנו ננתח את התוצאות על לוחות המשחקים ב 9X9 , כי שם התוצאות יותר ברורות וחד משמעיות, כמעט כל קבצי ההרצה עובדים יחסית מהר ב 7X7 ו 5X5. אבל המסקנות שנסיק עבור המשחקים של 9X9 תקפים עבור גדלים אחרים.

•כאשר הרצנו Backtracking רגיל , הוא פתר במהירות גבוהה יחסית את כל הלוחות חוץ מהאחרון (game5) בו הוא נתקע הרבה מאוד זמן(יותר מ 350 שניות), בעוד שקבצי ההרצה עם היוריסטיקות למיניהן לא הצליחו לפתור את ארבעה הלוחות הראשונים באותה מהירות של ה Backtracking הרגיל, אבל הקבצים עם היוריסטיקות פתרו את הלוח החמישי (game5) בזמן מצויין(פחות מ 5 שניות), במיוחד קובץ היוריסטיקות המשופר ביותר (MRVHD_LCV_FC9X9.py) שפתר את הלוח ב 0.37 שניות.

•נסתכל לרגע על הלוח הנ"ל ונשים לב לתכונה עיקרית בו שגורמת לביצועים טובים מאוד בקובץ הרצת היוריסטיקות האחרון, בעוד שב Backtracking הרגיל הוא 'נתקע'.



•בולט פה הוא כמות הרבה של המשבצות הריקות במצב ההתחלתי בשורות הראשונות בלוח. האלגוריתם של ה backtracking הרגיל לא עושה איזשהו variable ordering מיוחד, הוא מתקדם בלוח לפי סדר הופעתם של המשבצות בלוח (משמאל לימין-מלמעלה למטה). כלומר במשבצת הראשונה הוא יתחיל בלנסות ערכים מה דומיין של המשבצת בלי שום אילוץ שידוע לו מהמצב ההתחלתי , לכן סביר להניח שיעבור על מספר רב של ערכים עד מציאת ההצבה הנכונה בשורות ריקות אילו. בעוד שבקבצים

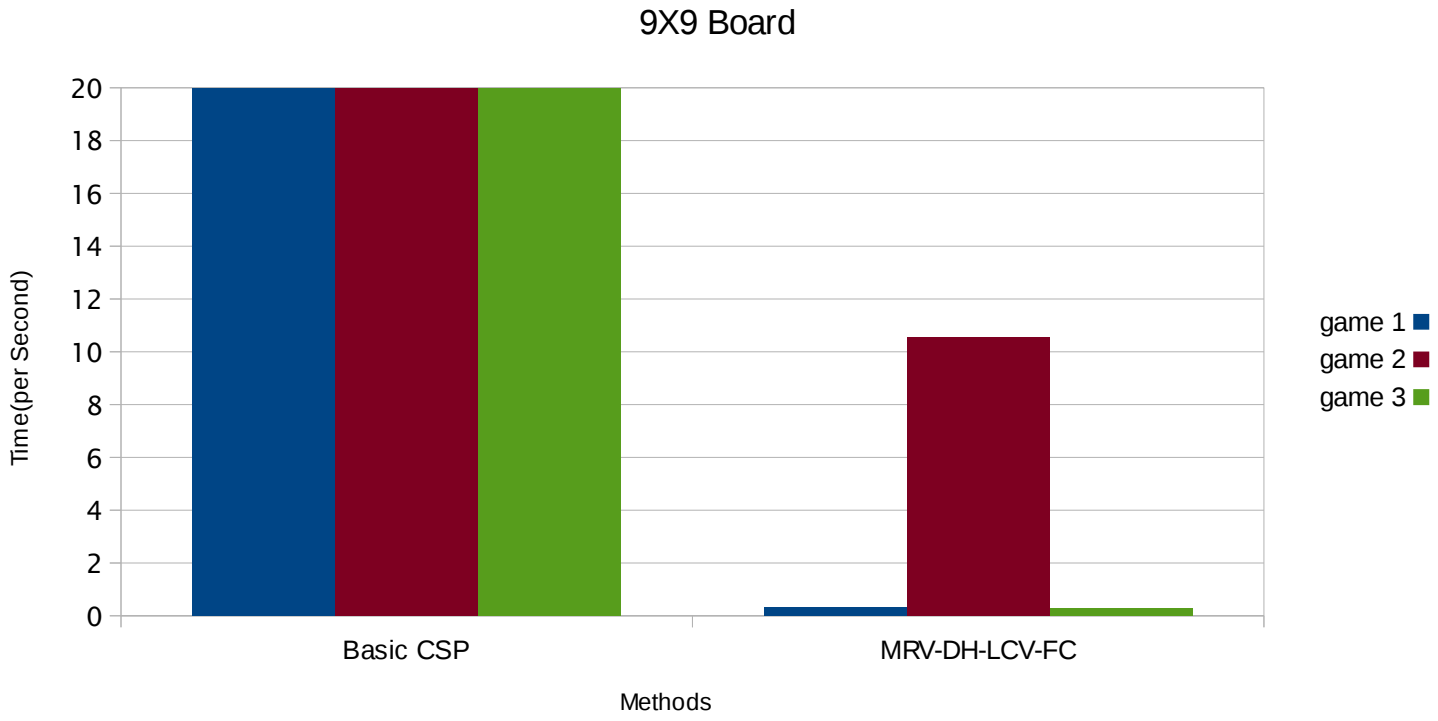
עם היוריסטיקה של MRV, המשבצות ימויינו בהתחלה לפי גודל ה domain שלהם , וההשמה תתחיל תמיד מאיזשהי משבצת מקור עם צבע מסויים (ה domain שלה הוא לכל היותר 4) שתשים יותר אילוצים רלוונטיים על שכנותיה כי אף פעם לא נשנה צבע של מקור(והצבת ערך השכן הבא תכפה צבע על אותו שכן), ואז האלגוריתם יתקדם מאזור עם 'צפיפות השמות' גבוהה לאיזור עם 'צפיפות השמות' נמוכה, ולכן ינסה פחות ערכים עד שיצליח.

•בלוחות האחרים בהן צפיפות ההשמות בלוח ההתחלתי היא גבוהה בחלק העליון של הלוח (לוחות 1-4 ב 9X9) , אין יתרון לקבצי ה MRV , והוא עושה מיונים מיותרים בסיבוכיות של(O (N^4 במספר אקספוננצלי בגודל הקלט ,בעוד שה Backtracking הרגיל יגיע לאותן תוצאות בלי מיונים(האילוצים בשורות הראשונות יצמצמו את ה domain במשבצות הריקות די מהר) ולכן ימצא הצבה נכונה מהר יותר.

•מצליחים לראות שה Forward Checking מייעל באופן ברור את החיפוש כך שמצמצם את ה BackTracking. בודק קדימה אם משתנה אי אפשר למלאות אותו.

•LCV ייעלה יותר מ MRV כצפוי לפני הרצת הבדיקות , כאן בבעיה ordering value יש לו משמעות .

- להלן עוד בדיקות שמאששות את המסקנות שלנו עבור איזה לוחות לא נפתרים בזמן סביר ב backtracking הבסיסי , מצאנו עוד לוחות המאופיינים בכמות הרבה של המשבצות הריקות במצב ההתחלתי בשורות הראשונות בלוח.
במקביל הפעלנו את קובץ ההרצה הכי טוב עבור מקרים אלה (שכולל את כל היוריסטיקות) ואלה התוצאות:



<u>הערה</u>

בגרף כאשר הזמן של משחק מסומן 20 שניות , הוא אפילו יותר מזה אבל כדי שהתוצאות הקטנות יהיו יותר ברורות .

הרצה ל-3 המשחקים הנ"ל ביוריסטיקות הנ"ל

python3 badbasic9X9.py
python3 goodMRVHD_LCV_FC9X9.py

Testing

ייצרנו 5 משחקים לכל אחד מהשלבים 5X5, 7X7, 9X9. וייצרנו קבצים להרצת כל היוריסטיקה על כל שלב.

כבודק

אפשר להריץ כל היוריסטיקה על כל שלב נתון באמצעות הקריאות הבאות:

N =5 ,7,9

python3 cspbasicNXN.py

להרצת BackTracking רגיל

python3 cspMRVNXN.py

להרצת Minimum Remaining Values

python3 cspMRVHNXN.py

להרצת Minimum Remaining Values + special Heuristic

python3 cspMRVHDNXN.py

#להרצת Minimum Remaining Values + special Heuristic + Degree Heuristic

python3 cspMRVHD_LCVNXN.py

להרצת Minimum Remaining Values + special Heuristic + Degree Heuristic +Least Constraining Values

python3 cspMRVHD_LCV_FCNXN.py

להרצת Minimum Remaining Values + special Heuristic + Degree Heuristic +Least Constraining Values + Forward Checking

הכנסת לוחות כ Input

כדי להכניס לוח מסויים NXN, יש להוסיף את הלוח לקובץ Tests.py ,כמטריצה דו מימדית ולהוסיף לה במקומות המתאימים את הצבעים (למשל 'B' – מסמן Blue ,
'0' – מסמן משבצת ללא צבע) ואחר כך להגדיר מערך מכיל את האותיות שמסמנים את הצבעים ששומשו בלוח. ולהוסיף את ה game = (colors, board) ל allNXNgames.

התוצאות ב SHELL

דוגמא להסבר

board to solve

Z 0 0 0 0 0 0 0 0
C 0 0 0 0 0 0 0 0
T 0 0 0 0 0 T 0 0
B 0 0 0 0 0 Y 0 0
Y 0 0 0 0 0 0 0 0
R 0 0 G 0 B 0 0 0
P 0 0 0 0 0 0 0 0
G P Z 0 C 0 0 0 0
R 0 0 0 0 0 0 0 0

----------solution----------

Z Z Z Z Z Z Z Z Z
C C C C C C C C Z
T T T T T T T C Z
B B B B B B Y C Z
Y Y Y Y Y B Y C Z
R G G G Y B Y C Z
R G P P Y Y Y C Z
R G P Z Z C C C Z
R R R R Z Z Z Z Z

solution found after 0.00810551643371582 seconds

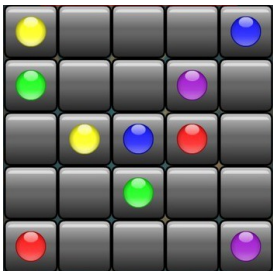
כל אות לועזית בלוח מסמל לצבע מסויים . אם זה '0' אזי אין צבע במשבצת ההיא.
הלוח הראשון הוא מה שצריך לפתור, השני לוח פתור, ו גם נתון הזמן שפותר בו הלוח בסוף.

APPENDIX

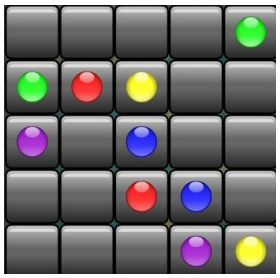
הלוחות שעבורם הורצו הבדיקות:

5X5

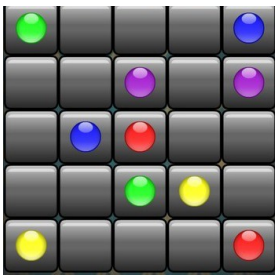
game1



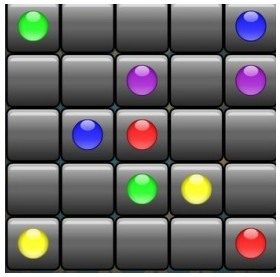
game2



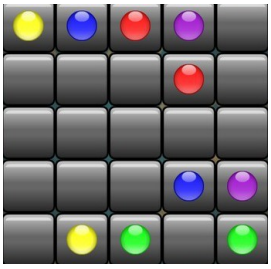
game3



game4

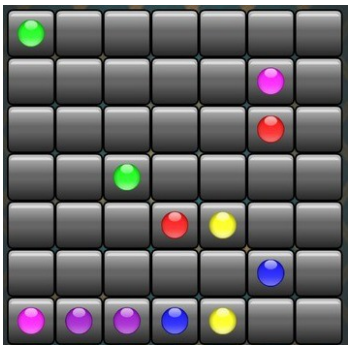


game5

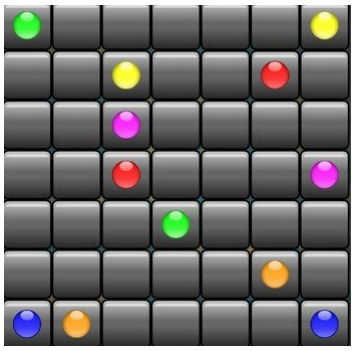


7X7

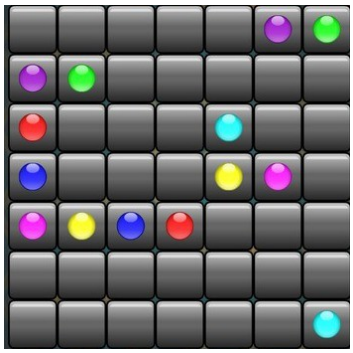
game1



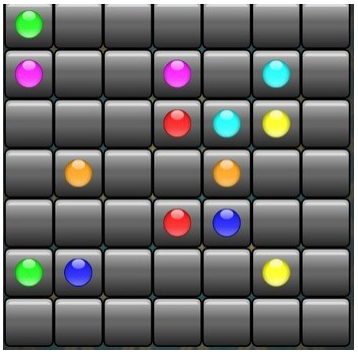
game2



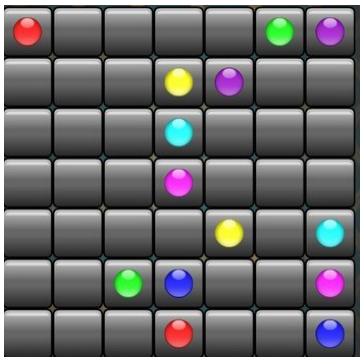
game3



game4

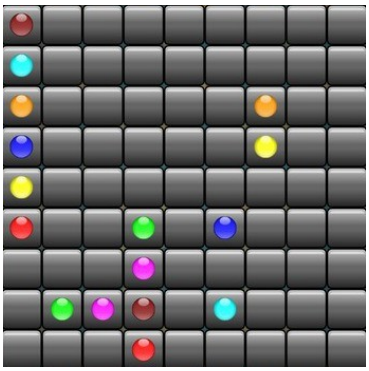


game5

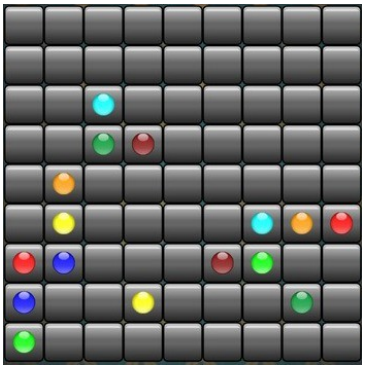


9X9

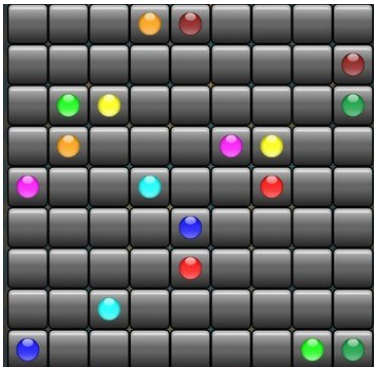
game1



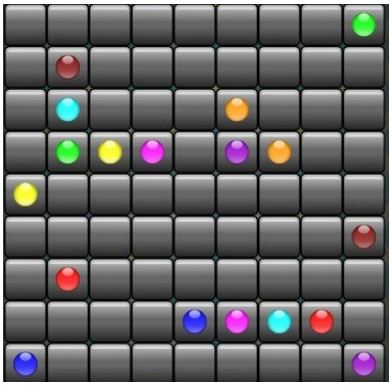
game5



game 2



game3



game4