2048

מרכז משחק וסימולציה מבוזר וממוקבל ב Erlang

פרויקט גמר קורס

Ben-Gurion University of the Negev
Faculty of Engineering Science - Dept. of Computer Engineering
Functional Programming in Concurrent and Distributed Systems
381-1-0112
Maor Assayag 318550746 | Refael Shetrit 204654891
2019

Hebrew עברית

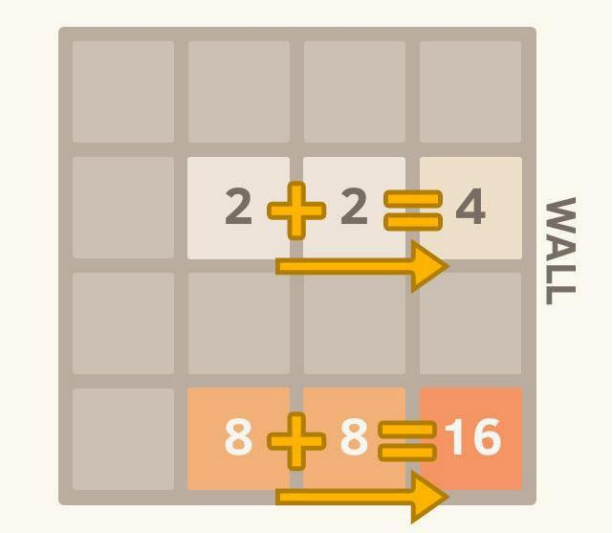
תוכן עניינים

תיאור המערכת

רקע

2048 הוא משחק רשת פשוט וממכר, עם חוקים פשוטים: הלוח כולל 16 משבצות, ועליהן ריבועים צבעוניים עם מספרים זוגיים. הפקדים הם 4 החיצים במקלדת הגורמים להזיז הריבועים במידת האפשר ימינה, שמאלה, למעלה או למטה. לאחר כל הזזה מופיע ריבוע חדש על המסך.

כשהשחקן דוחף שני ריבועים סמוכים עם אותו ערך, למשל 8 ו-8, הם מתחברים והופכים לסכום המספרים - 16 במקרה הזה. המטרה היא להגיע לריבוע של 2048. אם כל המשבצות בלוח תפוסות ואין יותר לאן לזוז, נפסלת וצריך להתחיל משחק חדש. סיבוב או שניים מספיקים כדי להבין את העיקרון, וניתן לראות הדגמה חיה של המשחק בלחיצה כאן.



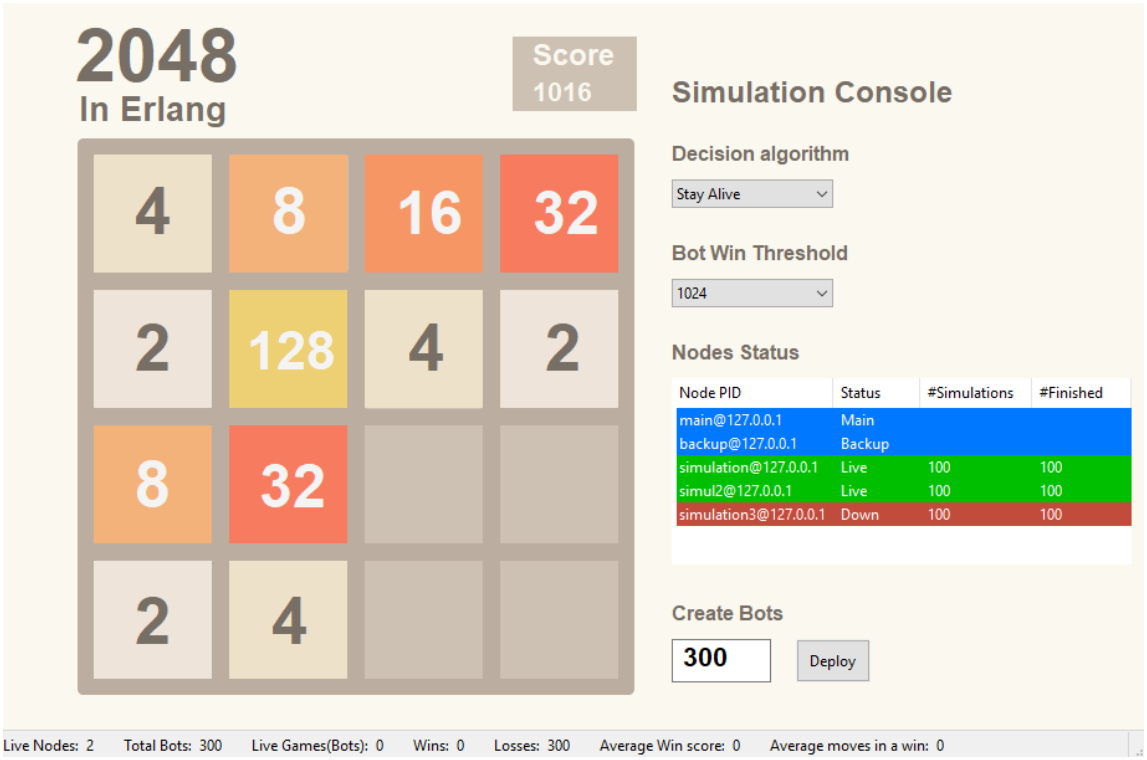
רעיון כללי

המערכת נחלקת ל2 : מרכז סימולציות מבוזר וממוקבל ומשחק משתמש ממוקבל עם תגובה בזמן אמת.

מרכז הסימולציות מאפשר הרצת סימולציות (בוטים) של משחקים ברקע על פי הגדרת מאפייני ריצה בממשק המשתמש (אלגוריתם בחירה לבוט, ערך רצוי של משבצת עבור הניצחון ומספר הריצות הרצוי) וקבלת סטטיסטיקות בלייב מכלל השרתים המחוברים לשרת הראשי. למערכת יש הגנה בפני נפילות, כך שבמידה ושרת סימולציה נופל ולא מספיק לספק סטטיסטיקות על כלל ההרצות הנוכחיות שהוא התבקש לבצע, המשימה שנותרה לו גם היא תבוזר לשאר שרתי הסימולציות.

בנוסף מתוחזק שרת גיבוי שמאפשר את היכולת של המערכת להתאושש באופן אוטומטי מנפילת השרת הראשי ובחירת אחד משרתי הסימולציה לתפקד כשרת ראשי. במהלך הסימולציות המידע מגובה באופן תדיר בשרת הגיבוי, כך ששרת ראשי חדש נוצר הוא לא מאבד את המידע שאספנו עד כה.

המשחק בזמן אמת מתוחזק על ידי 2 שרתים (שרת ממשק המשתמש והשרת הראשי עם תהליך ייעודי לניהול המשחק), כאשר כל קובייה מהווה תהליך המחזיק ערך ומיקום. המערכת בנויה ממנגנון פירמידת תהליכים (עליו נרחיב בהמשך) לשם איסוף ותחזוקת המידע על הלוח.



מבנה המערכת

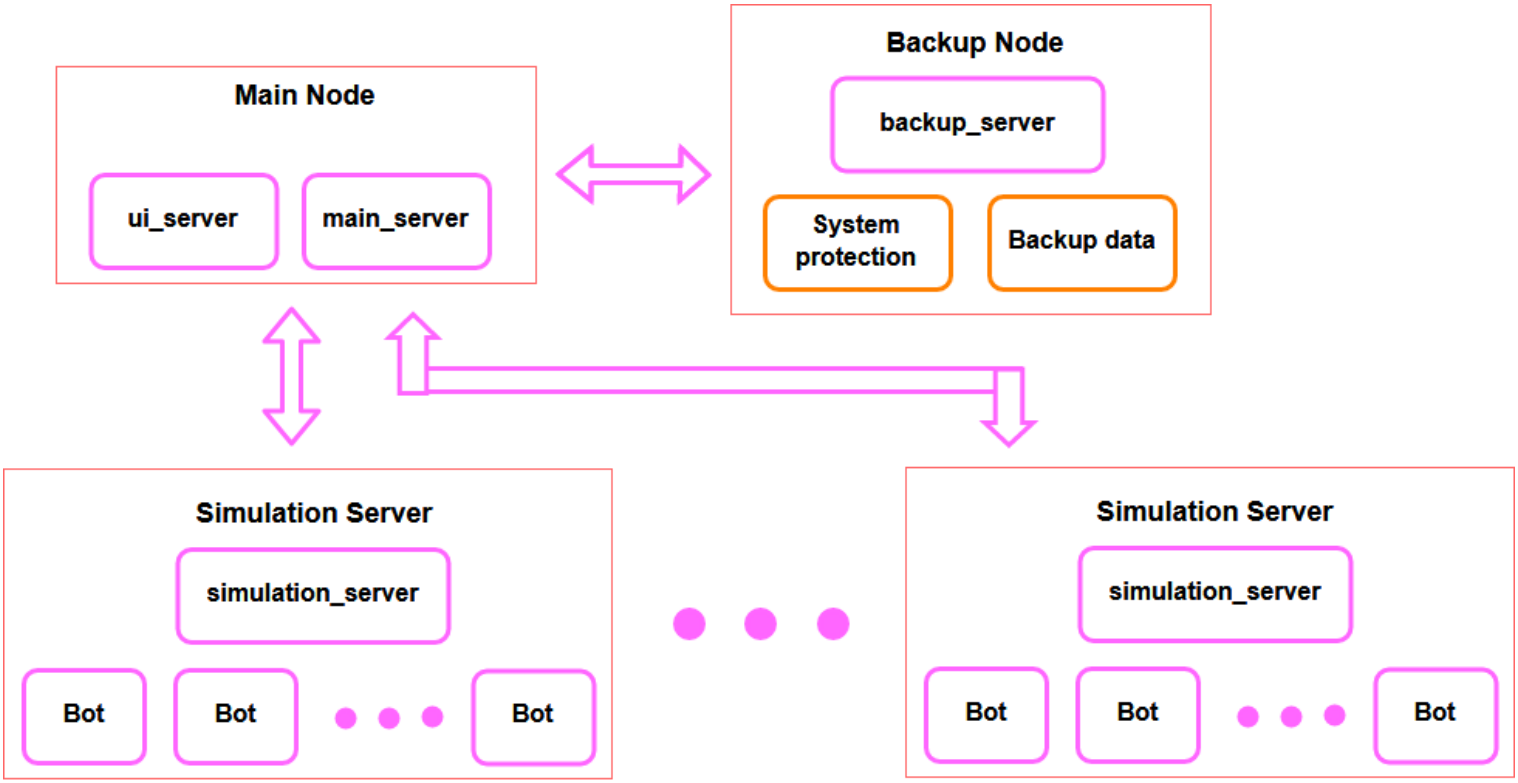
המערכת פותחה בארלנג (Erlang 21.0), פרוסה על כ 2500 שורות קוד ונחלקת ל 3 ישויות מרכזיות :

1. **מחשב ראשי** (יחיד בנקודת זמן) – בו ירוץ השרת המרכזי (main server) ושרת ממשק המשתמש (ui_server).
   - ניהול התצוגה הגרפית והממשק למשתמש – משחק בזמן אמת, קונסולת סימולציה וסטטסטיקה בזמן אמת.
   - אחריות על תקשורת והעברת נתונים בין שרתי הסימולצייה (משניים) לבין ממשק המשתמש וכן גיבוי בשרת הגיבוי.
   - אחריות על ניהול התהליכים, מקבול והלוגיקה של המשחק בזמן אמת.
   - ניטור השרתים המשניים ושרת הגיבוי בכדי להתגונן בפני נפילות להמשך פעילות תקינה.

2. **מחשב משני** (מרובה) – עליו ירוץ שרת סימולציה (simulation server) שמתחבר לשרת הראשי המנטר אותו (connect).
   - קבלת בקשות באופן רציף לכמות סימולציות רצויה, עבור כל משחק נפתח תהליך שמסמלץ בוט בהתאם לפרמטרי הסימולציה.
   - עדכון פרמטרי הסימולציה המחליטים על כיצד הבוט יבחר את המהלך הבא – מאלגוריתמים נאיביים (מהלכים רנדומליים, ניקוד מקסימלי) ועד אלגוריתמים המשומשים בלמידת מכונה (יורחב בהמשך).
   - שליחת מענה אסינכרוני לשרת הראשי עם תוצאות ביניים על סטטיסטיקת הסימולציות לשם עדכון התצוגה הגראפית.
   - במידת הצורך מחשב משני יהרוג את שרת הסימולציה ויהפוך לשרת הראשי תוך כדי שמירה על הנתונים שנאספו.

3. **מחשב גיבוי** (יחיד) – עליו ירוץ שרת הגיבוי (backup server), אליו מתחבר השרת הראשי (main server).

- תחזוקת בסיס נתונים מבוסס ets עבור כלל נתונים הסימולציה (יש לשמור עבור כל קומבינציית פרמטרים את הסטטיסטיקה שנאספה עד כה).
- קבלת עדכונים תדירים מהשרת הראשי ועדכון בסיס הנתונים.
- ניטור השרת הראשי והגנה בפני נפילות – במידה והשרת הראשי מחשב הגיבוי ייזום בקשת הקמת שרת ראשי חדש מאחד משרתי הסימולציה.
- אם שרת ראשי עולה מחדש, יש לבצע שחזור הנתונים מהגיבוי באופן אוטומטי עבור המשתמש (וכן בממשק המשתמש).



ui_server : wx_object,

simulation_server/backup_server/main_server : gen_server

שרת ראשי

Main server

מאפיינים

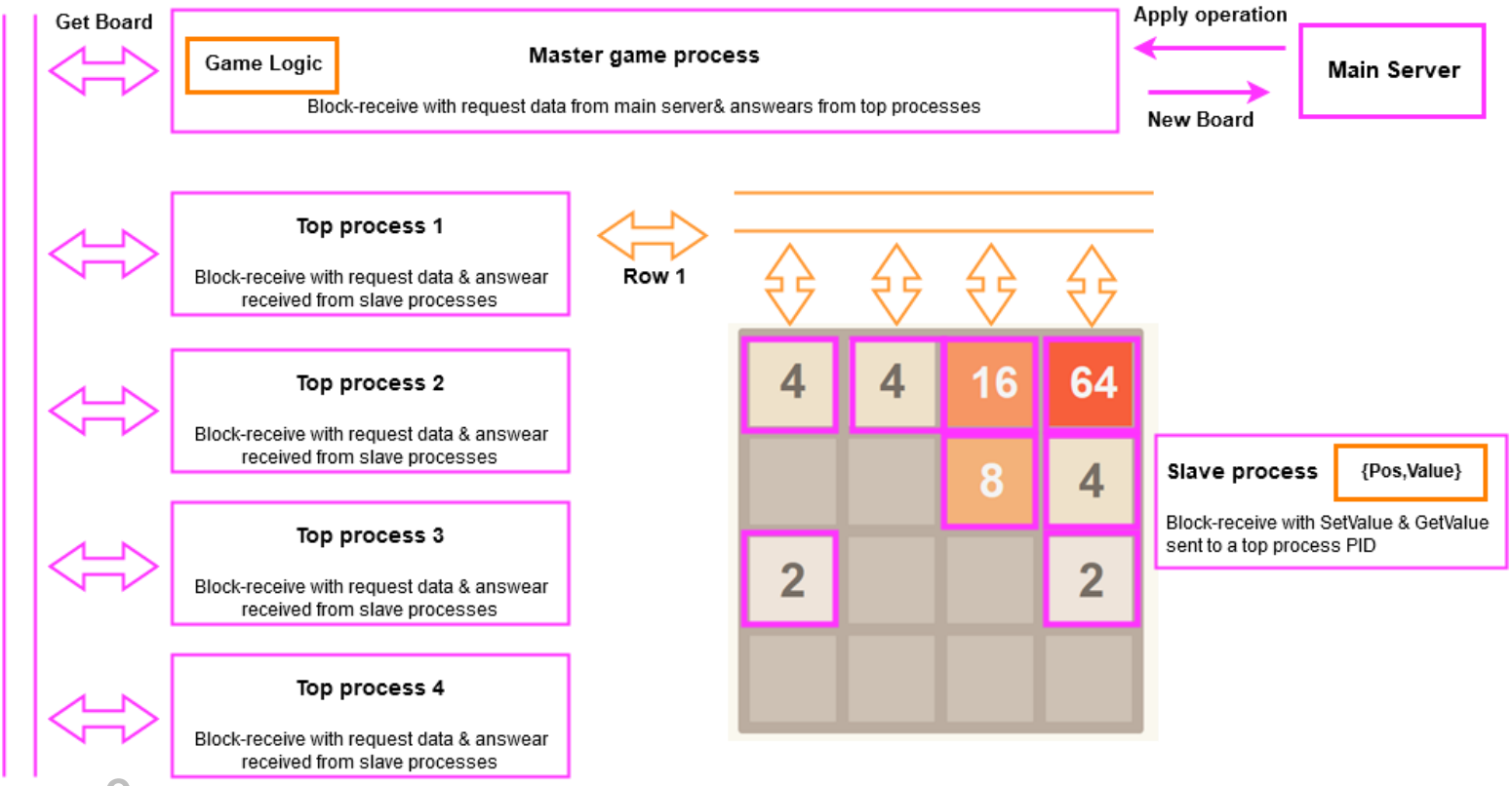
- מהווה את השרת המרכזי (**main_server**) שמתקשר ישירות עם ממשק המשתמש (**ui_server**).
- בקשות לתצוגה הגרפית והממשק למשתמש – משחק בזמן אמת, קונסולת סימולציה וסטטסטיקה בזמן אמת.
- אחריות על תקשורת והעברת נתונים בין שרתי הסימולצייה (משניים) לבין ממשק המשתמש וכן גיבוי בשרת הגיבוי.
- אחריות על ניהול התהליכים, מקבול והלוגיקה של המשחק בזמן אמת.
- ניטור השרתים המשניים ושרת הגיבוי בכדי להתגונן בפני נפילות להמשך פעילות תקינה.
- ממומש באמצעות **gen_server**
- במידה ושרת סימולציה נופל וסיפק רק מידע חלקי (כלומר חסרה סטטסטיקה על סימולציות שלא בוצעות) – אזי השרת הראשי יבזר את המשימות שנותרו ויוסיף לשרתים המשניים הנותרים בזמן אמת.
- תחזוקת מצב המשחק שמבצע המשתמש בלייב – כולל חישוב מצב המשחק החדש, האם המשחק נגמר, האם המשתמש הפסיד או ניצח (ניתוח מבני הנתונים בו שמור המצב החדש של המשחק).
- את המשחק שהמשתמש מבצע בלייב החלטנו למקבל לתקשורת בין תהליכים, ראה מקבול משחק.

טיפול בהודעות

אופי ההודעה	מוען	סוג ההודעה	מטרת ההודעה
connect	Any simulation_server	call	מעקב אחר השרתים המשניים (מוניטור ושמירת node(), pid) עדכון ממשק המשתמש עם פרטי שרת הסימולציה החדש
Request bots	ui_server	cast	ממשק משתמש מבקש לבזר x סימולציות עם פרמטרי סימולציה רצויים
update_stats	Any simulation_server	cast	עדכון סטטיסטיקה עבור פרמטרים מסויימים משרת סימולציה משני, מעדכן את ממשק המשתמש עם cast
update_simulation_params	ui_server	cast	עדכון פרמטרי הסימולציה עבור בקשות עתידיות, גיבוי בסיס הנתונים הנוכחי בשרת הגיבוי
update_user_game	main_server 'Top master process'	cast	לאחר שתהליך המשחק הראשי מסיים לחשב את הלוח החדש, השרת הראשי מקבל עדכון להעברה לשרת הממשק
game_move	ui_server	cast	ממשק המשתמש מיידע את השרת הראשי על מהלך של המשתמש (חיצי המקלדת), השרת הראשי מיידע את תהליך המשחק לאיסוף הלוח בתקשורת עם התהליכים המשניים
reset_game	ui_server	cast	איפוס המידע לגבי המשחק הנוכחי והתחלת משחק חדש בלחיצה על מקש הרווח במקלדת
deploy_backup	backup_server	cast	לאחר הרמת שרת ראשי מחדש, שרת הגיבוי מעדכן את השרת הראשי עם מסד הנתונים (ets)
nodedown	simulation_server backup_server	info	בעת נפילת שרת – הגנה בפני נפילות, עדכון ממשק המשתמש והמשך עבודה תקינה

ארכיטקטורת מקבול משחק משתמש

- תהליך **master** המנהל את המשחק ואת התקשורת מול כל תתי התהליכים, עובד במקביל לשרת **main_server** ומקבל ממנו בקשות ישירות עם המהלך שהמשתמש ביצע. תפקיד תהליך ה **master** לספק לשרת הראשי את הלוח החדש לאחר המהלך, מצב המשחק (נגמר\ניצחון או הפסד) וניקוד נוכחי.
- השרת הראשי מחזיק מבנה נתונים שעוקב אחר תתי התהליכים שמרכיבים את הלוח, הוא היחידי שייצור או יהרוס אותם.
- 4 תהליכים **top** שאחראים לתקשר עם מקסימום 4 תהליכי **slave** כל אחד. כל תהליך **slave** מחזיק ערך ומיקום, ויכול לשנות אותו או לשלוח אותו ל pid המצורף בבקשה.
- בקבלת מהלך חדש שרת ה **master** יבקש מ 4 שרתי **top** לספק את 4 השורות של הלוח. כל תהליך **top** יקבל עם הבקשה רשימה של התהליכים בשורה שלו, ועליו להחזיר תשובה ל **master** לאחר שירכיב את ערכי השורה שלו.
- בסיום איסוף הלוח תהליך ה **master** יפעיל את לוגיקת המשחק עם המהלך הרצוי על הלוח הנוכחי, יפעיל אנליזה לבדיקת סיום המשחק\ניצחון וניקוד חדש וישלח את התוצאות עם הלוח החדש ל **main server**.
- ה **main server** ייצור תהליכים חדשים \ יהרוג תהליכים שהמשבצת שלהם כבר לא מופיעה ויעדכן את ה-process board ב-record שב-state.



מבני נתונים

- השרת הראשי מתחזק State שהינו record שהגדרנו באופן הבא :
הערה : עבור כל קומבינציית פרמטרי סימולציה קיים record כזה שמתוחזק ב ets, וכל פעם נטען במידת הצורך ל State הנוכחי

ui_server_pid	מספר תהליך שרת ממשק המשתמש
totalBots	מספר סימולציות שהתבקשנו לבצע עד כה
liveBots	מספר סימולציות שרצות כרגע בשרתי הסימלוציות
totalWins	סך כל הנצחונות
totalLosts	סך כל ההפסדים
avgScoreWins	ניקוד ממוצע עבור משחקים שהסתיימו בניצחון
avgMovesWins	מספר מהלכים ממוצע עבור משחקים שהסתיימו בניצחון
Nodes {pid,node()}	שרתי סימולציות שרצים, עבור כל שרת סימולציה נשמור
totalNodes	מספר שרתי הסימלוציות שרצים כרגע
bot_threshold	פרמטר סימולציה, מכריע מתי משחק נחשב כניצחון
bot_decisionID	פרמטר סימולציה, מכריע את התנהגות הבוט בבחירת מהלך הבא
downNodes	היסטוריית שרתי סימולציה שנפלו
boardProcesses tuple, {{pos,pid}..{pos,pid}}	שמתאר את רשימת התהליכים לכל משבצת במשחק
Score	ניקוד נוכחי של משחק המשתמש

- עבור כל הקומבינציות הבאות מתוחזק record ב **ets** הלוקאלי (ובשרת הגיבוי) והוא נטען ל state הנוכחי של ה main server בהתאם לבחירת המשתמש

Move Decision By	**Max tile value to win**
Rand	128, 256, 512, 1024, 2048
Max score	128, 256, 512, 1024, 2048
Stay alive	128, 256, 512, 1024, 2048
Max merged tiles	128, 256, 512, 1024, 2048
Heuristic	128, 256, 512, 1024, 2048

UI server

מאפיינים

- ממומש כ **wx_object**
- מהווה את השרת ממשק המשתמש (**ui_server**) שמתקשר ישירות עם השרת הראשי (**main_server**).
- ניהול התצוגה הגרפית והממשק למשתמש – משחק בזמן אמת, קונסולת סימולציה וסטטסטיקה בזמן אמת.
- זיהוי המקשים הנלחצים על-ידי המשתמש (מקשי החיצים ורווח), תקשורת עם השרת הראשי לבקשת תגובה
- ניהול אלמנטים גרפיים בממשק המשתמש (טבלאות שרתים והמצב שלהם, סטטיסטיקה, קביעת פרמטרי סימולציה , משחק המשתמש ועוד)

טיפול בהודעות / איוונטים

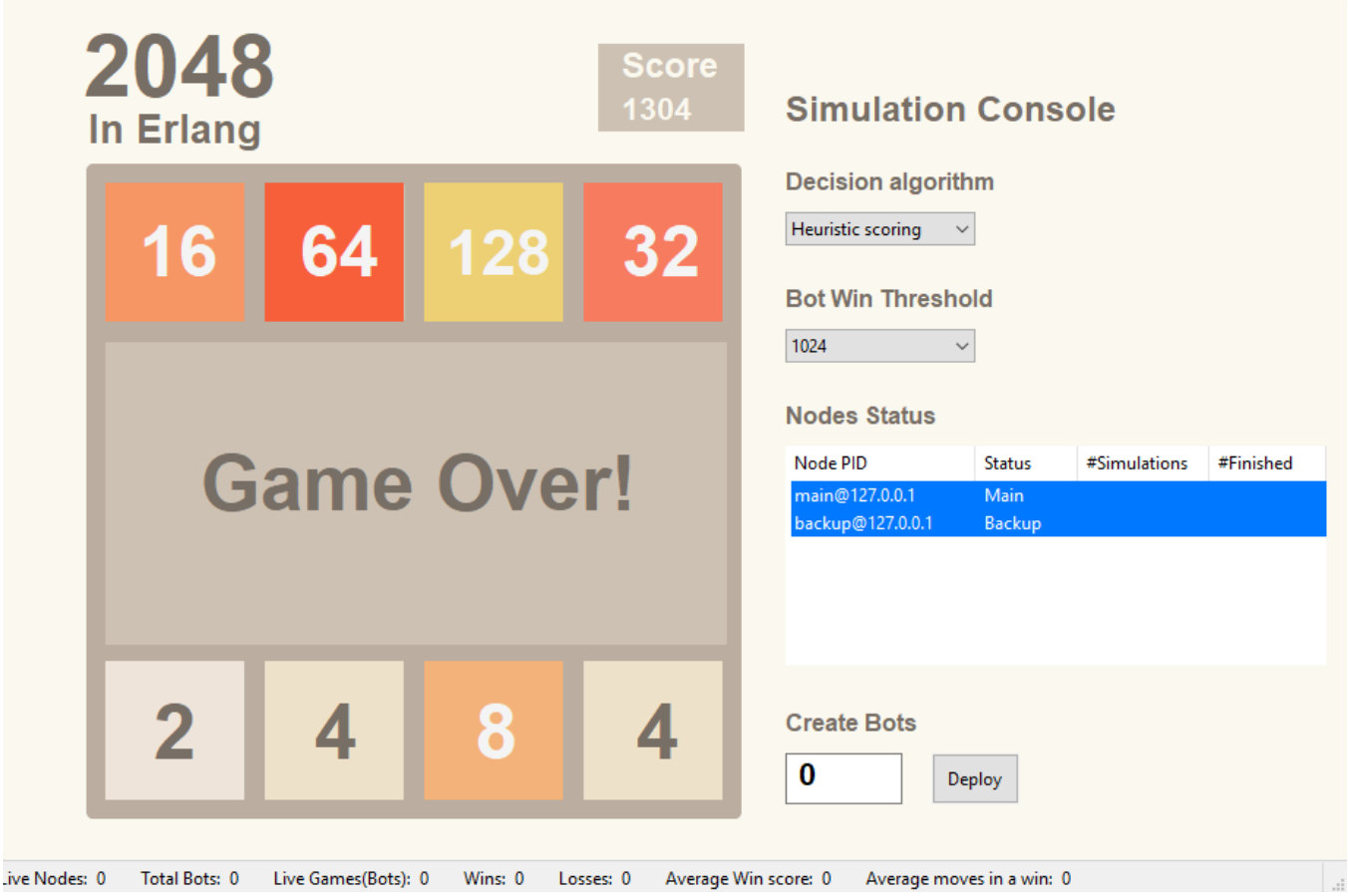
אופי ההודעה	מוען	סוג ההודעה	מטרת ההודעה
update_stats	main_server	cast	עדכון סטטיסטיקה (בר תחתון)
update_node_status	main_server	cast	עדכון מידע לגבי node בטבלאת הישויות בקונסלת הסימולציה
update_main_backup_nodes	main_server	cast	עדכון מידע לגבי node בטבלאת הישויות בקונסלת הסימולציה – לגבי שרת ראשי או שרת גיבוי
update_game	main_server	cast	עדכון משחק משתמש נוכחי
#wxKey{keyCode=?**SPACEBAR**}		event	איפוס משחק משתמש
#wxKey{keyCode=?**RIGHT**}		event	פעולת ימינה במשחק נשלח לשרת הראשי לביצוע
#wxKey{keyCode=?**LEFT**}		event	פעולת שמאלה במשחק נשלח לשרת הראשי לביצוע
#wxKey{keyCode=?**UP**}		event	פעולת למעלה במשחק נשלח לשרת הראשי לביצוע
#wxKey{keyCode=?**DOWN**}		event	פעולת למטה במשחק נשלח לשרת הראשי לביצוע

בקשת יצירת x סימולציות מהשרת הראשי עבור פרמטרי הסימולציה הנוכחיים	event		#wxCommand {type=command_choice_selected}
עדכון פרמטרי הסימולציה מ 2 תיבות בחירה	event		#wxCommand {type= command_choice_selected }

ממשק המשתמש

- אזור **שמאלי** – מהלך משחק בזמן אמת שנשלט בידי המשתמש, המשחק מתוחזק בשרת הראשי ע"י מקבול תהליכים (כל משבצת הינה תהליך כחלק מ"פירמידת" תהליכים המתקשרים עמם).
- אזור **ימני** – קונסולת סימולציית משחקים. מאפשרת למשתמש שליטה מלאה ביצירת x סימולציות (בוטים) על שרתי סימולציה שהתחברו לשרת הראשי (nodes אחרים). ניתן לשלוט בפרמטרי הסימולציה (סף ניצחון עבור בוט, שיטת בחירת הצעד הבא עבור הבוט)
- אזור **תחתון** – סטטוס סטטיסטיקה שמתעדכן בתדירות משרתי הסימולציה דרך השרת הראשי.

- טבלת ה nodes מציגה מעקב שמתעדכן בצורה מיידית לגבי השרתים השונים (השרת הראשי הנוכחי, שרת הגיבוי, שרתי הסימולציה) וסטטיסטיקה מתאימה.

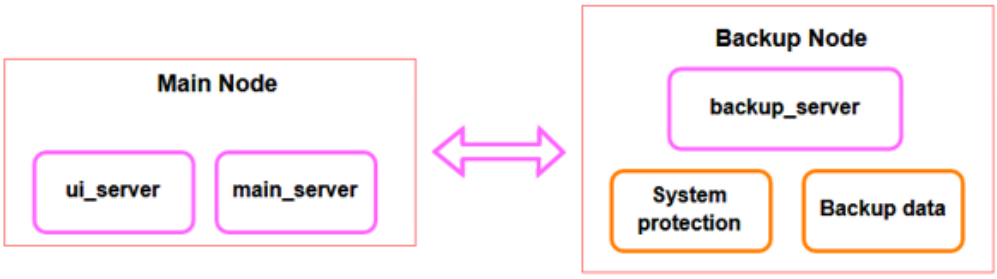


שרת גיבוי

Backup server

מאפיינים

- ממומש באמצעות **gen_server**
- תפקידו להוות גיבוי לכלל הנתונים שנאספו עד כה בסימולציות (עבור כל קומבינציית פרמטרי סימולציה), מתקשר ישירות עם השרת המרכזי (**main_server**) ובמקרה הצורך גם עם שרתי הסימולציה (**simulation_server**).
- ניטור השרת המרכזי בכדי להתגונן בפני נפילות להמשך פעילות תקינה – במידת והשרת המרכזי נופל, שרת הגיבוי יבחר את אחד משרתי הסימולציה ויבקש ממנו להפוך לשרת ראשי. המעבר לתצוגת ממשק משתמש חדשה (לאחר שהשרת המרכזי נפל) הינה מיידית.
- כששרת ראשי חדש עולה שרת הגיבוי ישלח לו הודעת עדכון עם כלל מסד הנתונים הנוכחי להמשך פעילות תקינה. השרת הראשי החדש מנטר גם כן את שרת הגיבוי בכדי שנוכל לעדכן את ממשק המשתמש בהתאם לסטטוס של שרת הגיבוי.



טיפול בהודעות

אופי ההודעה	מוען	סוג ההודעה	מטרת ההודעה
connect	main_server	call	מעקב אחר השרת המרכזי הנוכחי(מוניטור ושמירת pid ,()node) שליחת הודעת deploy backup לשרת הראשי עם הגיבוי
update_ets	main_server	cast	עדכון טבלת הנתונים עבור record ספיצפי (על-פי פרמטרי הסימולציה) עדכון מתבצע בעת החלפת פרמטרים
new_simulation_node	main_server	cast	עדכון על התחברות שרת סימולציה חדש. על שרת הגיבוי להכיר אותם בשביל הגנה בפני נפילות
simulation_node_down	main_server	cast	עדכון על סטטוס שרת סימולציה. על שרת הגיבוי להכיר אותם בשביל הגנה בפני נפילות
nodedown	main_server	cast	הגנה בפני נפילות, תחילת פרוצדורת בחירת שרת ראשי חדש

מבני נתונים

- השרת הראשי מתחזק State שהינו record שהגדרנו באופן הבא :
הערה : עבור כל קומבינציית פרמטרי סימולציה קיים record כזה שמתוחזק ב ets, וכל פעם נטען במידת הצורך ל State הנוכחי

ui_server_pid	מספר תהליך שרת ממשק המשתמש
totalBots	מספר סימולציות שהתבקשנו לבצע עד כה
liveBots	מספר סימולציות שרצות כרגע בשרתי הסימלוציות
totalWins	סך כל הנצחונות
totalLosts	סך כל ההפסדים
avgScoreWins	ניקוד ממוצע עבור משחקים שהסתיימו בניצחון
avgMovesWins	מספר מהלכים ממוצע עבור משחקים שהסתיימו בניצחון
Nodes {pid,node()}	שרתי סימולציות שרצים, עבור כל שרת סימולציה נשמור
totalNodes	מספר שרתי הסימלוציות שרצים כרגע
bot_threshold	פרמטר סימולציה, מכריע מתי משחק נחשב כניצחון
bot_decisionID	פרמטר סימולציה, מכריע את התנהגות הבוט בבחירת מהלך הבא
downNodes	היסטוריית שרתי סימולציה שנפלו
boardProcesses tuple, {{pos,pid}..{pos,pid}}	שמתאר את רשימת התהליכים לכל משבצת במשחק
Score	ניקוד נוכחי של משחק המשתמש

- עבור כל הקומבינציות הבאות מתוחזק record ב **ets** לוקאלי

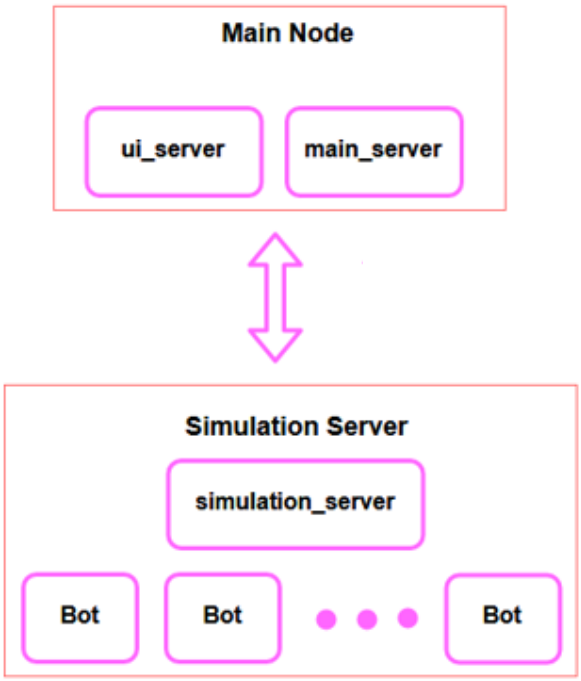
Move Decision By	**Max tile value to win**
Rand	128, 256, 512, 1024, 2048
Max score	128, 256, 512, 1024, 2048
Stay alive	128, 256, 512, 1024, 2048
Max merged tiles	128, 256, 512, 1024, 2048
Heuristic	128, 256, 512, 1024, 2048

שרת סימולציה

Simulation server

מאפיינים

- ממומש באמצעות **gen_server**
- מרובה, לא מוגבל במספר התחנות (Nodes) שיכולים להקים אותו ולהתחבר לשרת הראשי.
- כל שרת סימולציה ימקבל את מספר הסימולציות שהוא נדרש לבצע (לדוגמה נדרש לבצע 200 סימולציות של משחקים אז הוא יקים 200 תהליכי בוטים כשכל תהליך נדרש לסמלץ משחק).
- כל תהליך כזה יחשב מקומית את המשחק והמצב שלו בהתאם לפעולות (הבוט) שהוא מנהל.
- מאתחל טבלת ets שמשותפת לכל התהליכים בשרת – בכדי למנוע צוואר בקבוק של הודעות עדכון מהתהליכים השונים ניתן לעדכן ישירות במסד נתונים משותף את התוצאות (על-ידי פעולות אטומיות בספריית ets).
- כששרת ראשי נופל, שרת הגיבוי בוחר שרת סימולציה ומבקש ממנו להפוך לשרת ראשי במקום (כלומר לסגור את ה simulation server ולהקים main server).
- פרמטרי הסימולציה קובעים כיצד כל בוט ינהל את המשחק, הסבר מורחב בהמשך.



טיפול בהודעות

אופי ההודעה	מוען	סוג ההודעה	מטרת ההודעה
become_main	backup_server	call	הפיכת שרת הסימולציה הנוכחי לשרת הראשי כחלק מהגנה בפני נפילות והמשך פעילות תקינה
update_stats	simulation_server	cast	כל עוד יש עדיין סימולציות שלא הסתיימו מתבצע עדכון מחזורי אל השרת המרכזי עם סטטיסטיקות נוכחיות
request_bots	main_server	cast	בקשת הרצת x סימולציות עם פרמטרי סימולציה מצורפים
kill_simulation_server	simulation_server	cast	שרת הגיבוי מחכה לתגובה כשהוא מבקש לייצר את השרת הראשי, לכן ההריגה של שרת הסימולציה מתבצע רק לאחר קבלת cast עצמי

מבני נתונים

- שרת הסימולציה מתחזק State שהינו record שהגדרנו באופן הבא :

מספר תהליך שרת ממשק המשתמש	main_server_pid
מספר סימולציות שהתבקשנו לבצע עד כה	totalBots
סך כל הנצחונות	totalWins
סך כל ההפסדים	totalLosts

- כל בוט (תהליך) מתחזק את מצב המשחק שהינו record שהגדרנו באופן הבא :

ייצוג לוח 2048	board
האם המשחק נגמר	game_over
האם כרגע יש משבצות פנויות בלוח	freeTiles
ניקוד נוכחי	score
האם ניצח	won

- בנוסף מתוחזקת טבלת ets עבור הסטטיסטיקות (כך שכל תת תהליך יוכל לעדכן אותה בצורה אטומית ולמנוע צוואר בקבוק על שרת הסימולציה) :

Key	Default value	Purpose
threshold	2048	ערך משבצת לניצחון
wins	0	סך הניצחונות
losts	0	סך ההפסדים
score_wins	0	ניקוד ממוצע בנצחונות
moves_wins	0	מספר מהלכים ממוצע בניצחונות
update_stats	false	האם לעדכן את השרת המרכזי בפרמטרים

פרמטרי סימולציה

Threshold

ערך תת סף מאפשר לבוטים (סימולציות) לנצח במשחק. ללא למידת מכונה או לקיחת בחשבון חישוב לעומק (עבור כל צעד לחשב מספר צעדים קדימה), להגיע לערך 2048 במשבצת כדי לנצח הינה משימה מורכבת. לשם כך אפשרנו להגדיר את ערך תת הסף מחדש וכך לראות נתונים טיפה יותר מעניינים. אנו מאפשרים שינוי של ערך הסף ל [2048, 1024, 512, 256, 128].

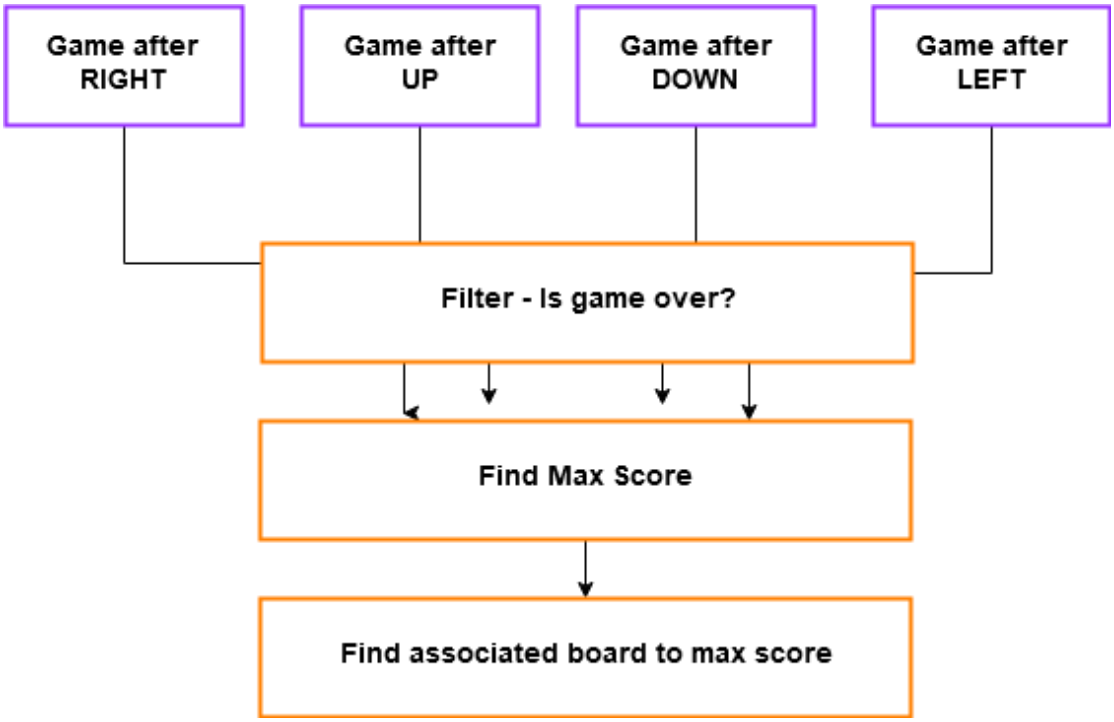
Decision ID

בכדי לאפשר גיוון בסימולציות אנו מאפשרים בחירה בין 5 אופני עבודה עבור התנהגות הבוטים כאשר הם נדרשים לבצע את השלב הבא :

- **Random** – הבוט מבצע מהלכים רנדומליים המתפלגים באופן אחיד [up, down, right, left]. סימולציה זו באופן טבעי היא הכי מהירה ומספר הניצחונות בה צפוי להיות הכי קטן מתוך מדגם גדול והשוואתו לשאר השיטות.

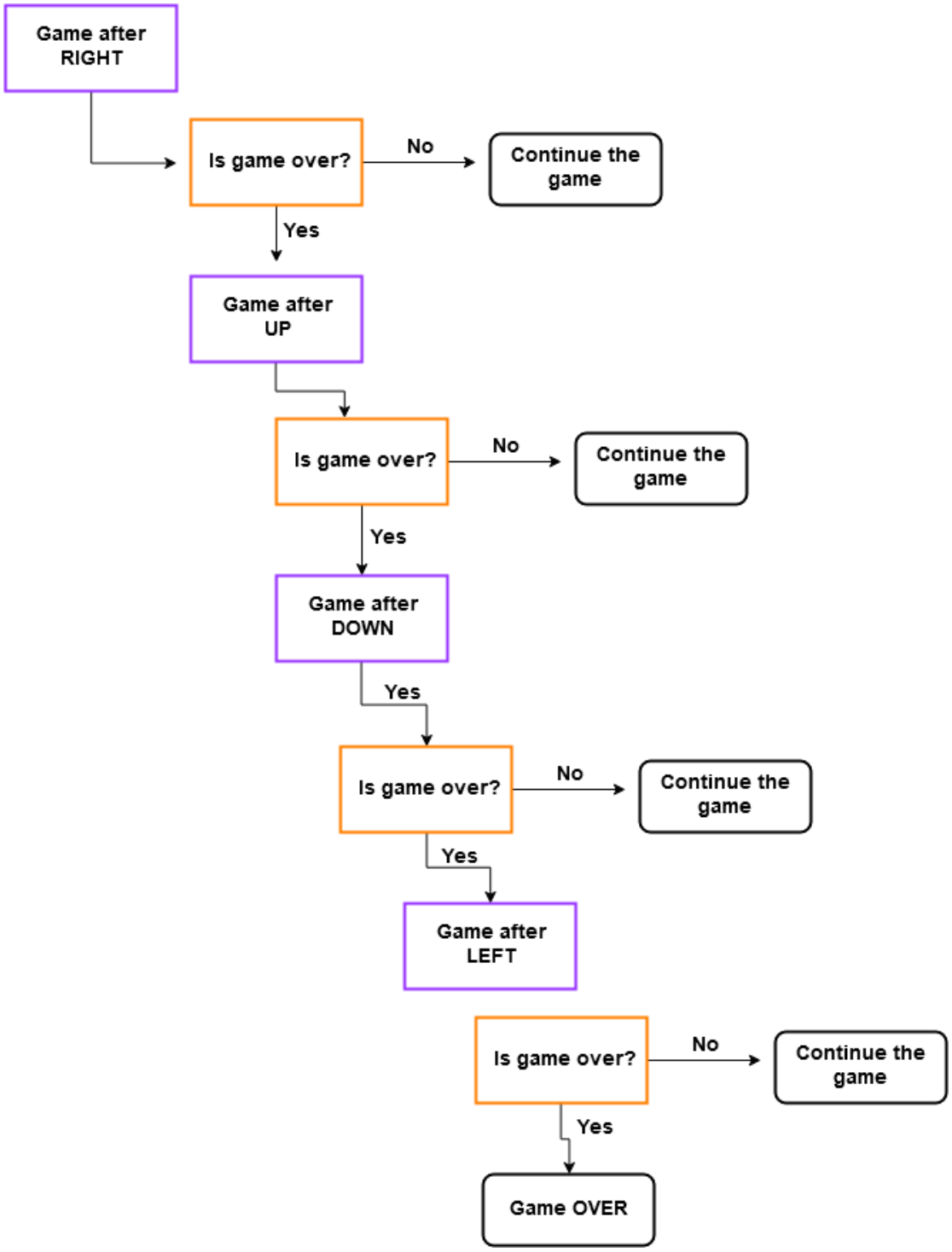
- **Max score** – הבוט מחשב את 4 התוצאות האפשריות לאחר ביצוע כל מהלך בנפרד [up, down, right, left]. על התוצאות אנו מבצעים **פילטר** למשחקים שלא נגמרו (כלומר שלא הפסדנו)**.** מתוך המשחקים הללו אנו בוחרים את המשחק עם הניקוד המקסימלי וממשיכים לצעד הבא. כמובן שאם כלל המהלכים מובילים למבוי סתום (המשחק נגמר) אזי הבוט יסכם את התוצאות.

עבור כל חישוב מהלך בנפרד יש לא רק לבצע את לוגיקת המשחק בעקבות המהלך הרצוי (לדוגמה הזזה ימינה), אלא גם לבצע אנליזה על מצב המשחק (הסתיים\ניצחון).



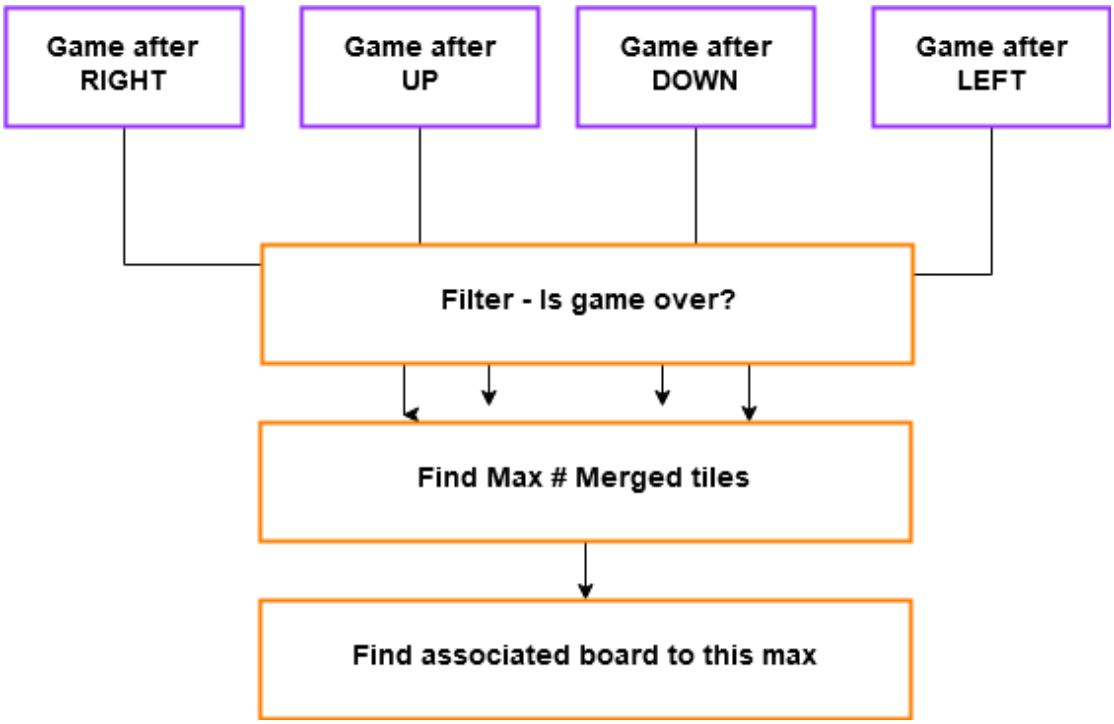
- **Stay Alive** – הבוט מחשב בצורה סדרתית מהלך, אם המהלך מוביל להפסד מידי נסה לחשב את תזוזה אחרת. כאשר המטרה היא לבצע כמה שיותר מהלכים (להישאר בחיים).

עבור כל חישוב מהלך בנפרד יש לא רק לבצע את לוגיקת המשחק בעקבות המהלך הרצוי (לדוגמה הזזה ימינה), אלא גם לבצע אנליזה על מצב המשחק (הסתיים\ניצחון).



- **Max Merged Tiles** – הבוט מחשב את 4 התוצאות האפשריות לאחר ביצוע כל מהלך בנפרד [up, down, right, left]. על התוצאות אנו מבצעים **פילטר** למשחקים שלא נגמרו (כלומר שלא הפסדנו). מתוך המשחקים הללו אנו בוחרים את המשחק עם **מספר מיזוגי המשבצות המקסימלי** וממשיכים לצעד הבא. כמובן שאם כלל המהלכים מובילים למבוי סתום (המשחק נגמר) אזי הבוט יסכם את התוצאות.

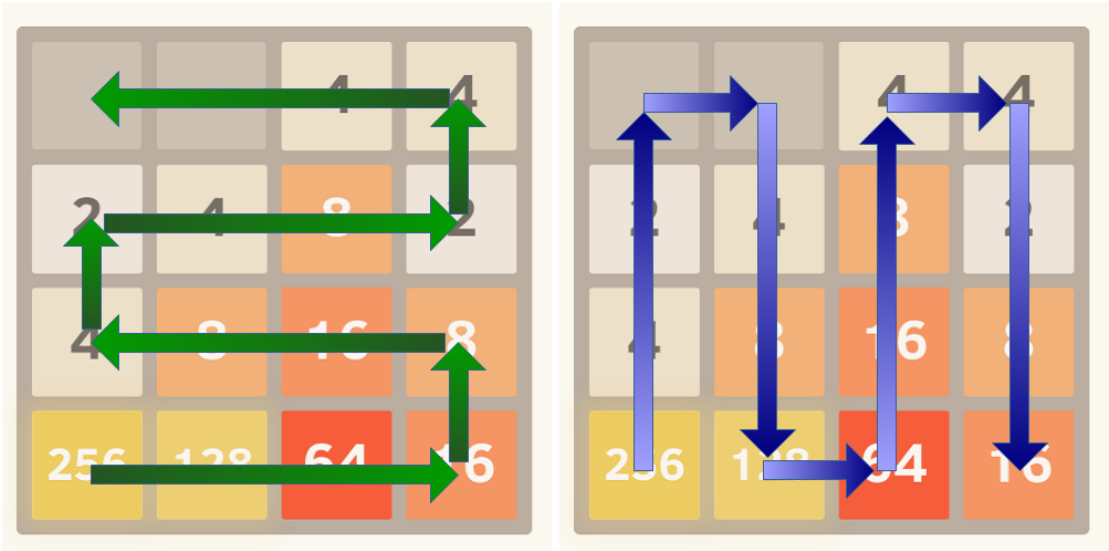
עבור כל חישוב מהלך בנפרד יש לא רק לבצע את לוגיקת המשחק בעקבות המהלך הרצוי (לדוגמה הזזה ימינה), אלא גם לבצע אנליזה על מצב המשחק (הסתיים\ניצחון).



- **Heuristic score** – הבוט מחשב את 4 התוצאות האפשריות לאחר ביצוע כל מהלך בנפרד [up, down, right, left]. על התוצאות אנו מבצעים אנליזה שהוכחה להיות האופטימלית ביותר מבחינת מספר ניצחונות עבור 2048.

הרעיון הכללי של השיטה הינה להעניק ניקוד למסלולים לינאריים בלוח (למשל ל 8 מסלולים) ולבחור את הניקוד המקסימלי מבין המסלולים כניקוד שמייצג את המהלך הנבחן. בדרך כלל מיישמים שיטה זו עם חישוב לעומק (כמה מהלכים קדימה), אומנם אנו לא ביצענו זאת מטעמי מטרת הפרויקט.

החלטנו להתמקד ב **2** מסלולים ליניאריים, כדלהלן :

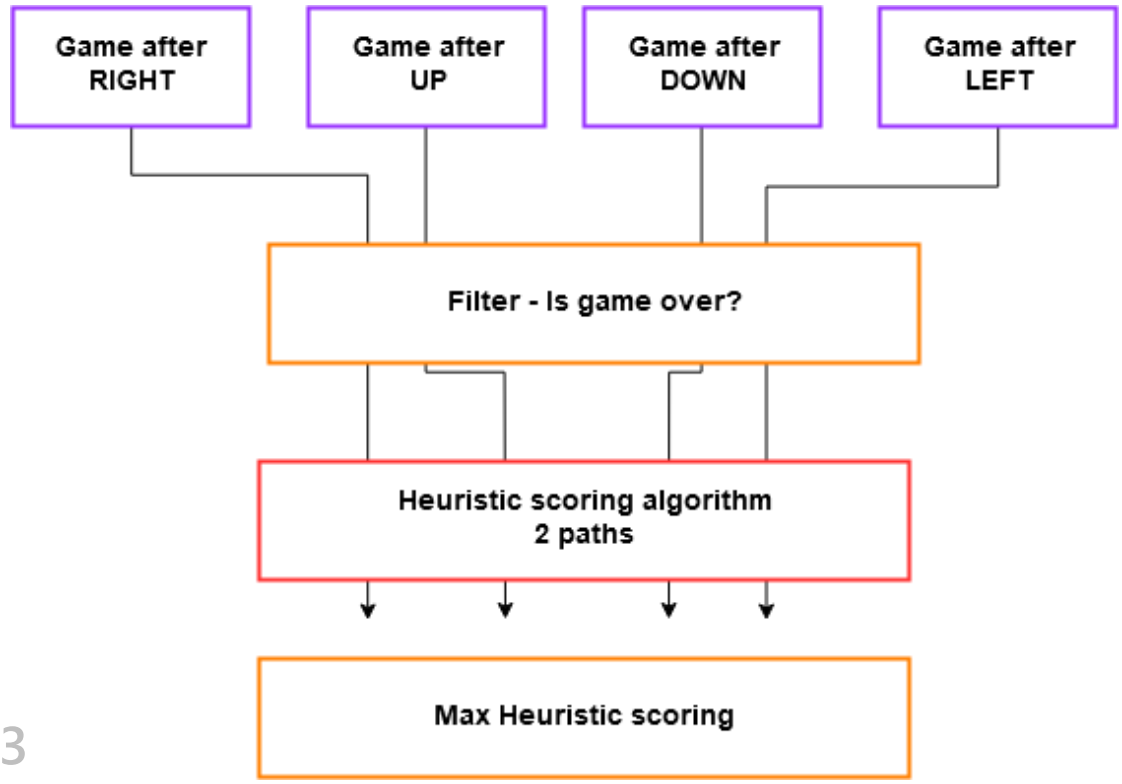


עבור כל מסלול אנו מחשבים את הניקוד בצורה הבאה :

$$p_n \in Path_{0 \cdots N-1}$$
$$score = \sum_{n=0}^{N-1} value(p_n) * r^n$$

כאשר בחרנו את r להיות **0.5** (אנו מסתמכים על אנליזה שפורסמה בקישור הבא).

עבור כל חישוב מהלך בנפרד יש לא רק לבצע את לוגיקת המשחק בעקבות המהלך הרצוי (לדוגמה הזזה ימינה), אלא גם לבצע אנליזה על מצב המשחק (הסתיים\ניצחון) וכן את אנליזת הניקוד. לבסוף נבחר המהלך עם הניקוד הגבוה ביותר.



בעיות ופתרונן

בעיה	השלכות	גישה לשיפור
צוואר בקבוק ב main server	• איטיות בתגובה לפעולת משתמש חדשה במשחק • עדכון לא בזמן את של סטטיסטיקה	• הפרדת שרת main server לשרת נוסף ui sever האחראי על עדכון התצוגה הגראפית וקבלת חיווי המשתמש • מקבול השרת הראשי (טיפול בתקשורת \ ארכיטיקטורת מקבול תחזוק משחק המשתמש)
צוואר בקבוק ב simulation server	• מהירות יכולה להשתפר • משוב סטטיסטיקה איטי	• כל תהליך מעדכן טבלת ets משותפת בצורה אטומית וככה נמנעים מהודעות עדכון cast מכל בוט • כל בוט מאותחל בתהליך משלו, בסיום הסימולציהההתהליך הורג את עצמו
חישוב לעומק עבור כל בוט (מספר מהלכים קדימה)	• לעיתים גרם לקריסת סימולציות עקב overhead במחשב האישי	• מחשבים מהלך אחד קדימה
הגנה בפני נפילות שרתי סימולציה	• לא מקבלים סטטיסטיקה מלאה במידה ושרת הסימולציה לא הספיק לסיים את כלל הסימולציות	• הגנה בפני נפילות – כשמזוהה נפילה עם שרת שנתן מענה חלקי העבודה שנותרה לו מתחלקת באופן שווה עם שרתי הסימולציות שנותרו
הגנה בפני נפילת השרת הראשי	• התוכנית מפסיקה לעבוד וכלל המידע שאספנו נאבד	• הקמת שרת גיבוי backup שיתחזק מסד נתונים השומר את כלל נתוני הסימולציה • בזיהוי נפילת שרת ראשי נשלחת בקשה משרת הגיבוי לאחד משרתי הסימולציה המחוברים להפוך לשרת הראשי, וכך באופן רציף נשמרת פעילות תקינה עם אי-איבוד מידע
מימוש נרחב עם wx_object הכולל תחזוקת משחק 2048, תחזוקת סטטיסטיקה, טבלאות בחירה ועדכון טבלאות תצוגה		• היעזרות במדריכי עזר מקורסים מקבילים באוניברסטאות בחו"ל ובעיקר שימוש בדוקומנטציית ארלנג וב **wx:demo()**

מסקנות

- בפיתוח מערכת חשוב לתכנן מראש את הארכיטקטורה בצורה מפורטת ככל הניתן, בעיקר כשמדובר בדינמיקה בין מספר שרתים וביזור\מקבול משימה.

- במהלך התכנון חשוב לשים לב לשלבים היכולים לגרום לצוואר בקבוק עקב תקשורת מרובה או העמסת משימות על תהליך. לעיתים הפתרון הוא פשוט למקבל למספר תהליכים באותו השרת את הפעולות השונות, או לפרק לשרתים עם ייעוד שונה שרצים על אותו המחשב.

- לעיתים עדיף להשתמש במסד נתונים משותף (כגון טבלת ets או mnesia) בכדי לאסוף נתונים ממספר מרובה מאוד של תהליכים (לדוגמה הרצת 10000 סימולציות של משחקים בכל שרת סימולציה יהווה צוואר בקבוק אם כל תהליך כזה יעדכן בהודעת cast את שרת הסימולציה שלו לגבי תוצאות המשחק).

- עבור תכנות מקבילי ומבוזר ארלנג מספקת כלים וסביבת עבודה לפיתוח כמה שיותר יעיל ומהיר, ולעיתים אף השימוש בה הינו האופטימלי בהתאם למשימה הנדרשת ובעיקר כשמדובר בתקשורת בין תחנות\תהליכים ומקבול למספר תהליכים רב.

2048

הוראות הפעלה

ההוראות הבאות מיועדות למערכת ההפעלה windows וללינוקס.

ההוראות מיועדות לעבודה על מחשב אחד (כך שאנו פותחים מספר nodes על אותו המחשב) או לעבודה עם מספר מחשבים המקונפגים ברשת התקשורת לתקשר אחד עם השני.

השתמשנו ב Erlang 21.0, המערכת תפעל בכל גרסה שתומכת ב OTP (לפחות גרסה 18).

שיתוף תקשורת בין מחשבים ברשת

ההוראות הבאות תקפות ללינוקס ubuntu (המחשבים שיש במעבדת הקורס).

נעשה ע"י עריכת קובץ hosts בכל אחד מהמחשבים הרצויים.

הקובץ נמצא ב /etc/hosts.

כדי לערוך קובץ זה יש להתחבר ל (root) super user על-ידי :

su ubu

password : local621 (the password for the ubu user)

sudo -s

password: local621 (or other password for the super user)

cat /etc/hosts

עבור כל מחשב חדש שרוצים להוסיף לרשת התקשורת יש להוסיף את השורה

ip hostname

כאשר את ה ip של המחשב אפשר לראות ע"י ifconfig ואת ה hostname ע"י hostname.

אם נדרשת, הסיסמא למשתמש cse_student הינה !123Qwe.

מחשב גיבוי

1. פתיחת טרמינל

2. פתיחת טרמינל erl עם שם ייחודי לשרת הגיבוי וכתובת ה ip של ה localhost
erl -name backup@127.0.0.1 -setcookie "abc"

ברשת תקשורת (לדוגמה במעבדה)
erl -sname backup -setcookie abc

3. עבור משתמשי ווינדוס בתוך הטרמינל של ה erl ננווט לתיקיית הפרויקט
cd("C:/Users/temp/Desktop/Erlang/my2048").
עבור משתמש לינוקס אפשר לפתוח את הטרמינל ישירות בתיקיית הקבצים

4. נקמפל את כלל הקבצים

c(backup_server).

c(main_server).

c(simulation_server).

c(ui_server).

עבור גרסאת לינוקס נקמפל גם את הקובץ הנוסף

c(ui_server2).

5. נפעיל ישירות את שרת הגיבוי

backup_server:start().

מחשב ראשי

1. פתיחת טרמינל

2. פתיחת טרמינל erl עם שם ייחודי לשרת הגיבוי וכתובת ה ip של ה localhost
erl -name main@127.0.0.1 -setcookie "abc"

   ברשת תקשורת (לדוגמה במעבדה)
   erl -sname main -setcookie abc

3. בתוך הטרמינל של ה erl ננווט לתיקיית הפרויקט
cd("C:/Users/temp/Desktop/Erlang/my2048").
עבור משתמש לינוקס אפשר לפתוח את הטרמינל ישירות בתיקיית הקבצים.

4. נקמפל את כלל הקבצים אם לא קמפלנו אותם עד כה (אפשר להפעיל ללא שרת גיבוי)
c(backup_server).
c(main_server).
c(simulation_server).
c(ui_server).
עבור גרסאת לינוקס נקמפל גם את הקובץ הנוסף
c(ui_server2).

5. נפעיל ישירות את השרת הראשי עם שם ה node שנתנו לשרת הגיבוי
main_server:start('backup@127.0.0.1').

   ברשת תקשורת (לדוגמה במעבדה)

   main_server:start('backup@hostname).

   כאשר hostname הינו שם ה node של ה backup ברשת, ניתן למצוא ע"י hostname בטרמינל.

   שרת ממשק המשתמש יפתח מיידית את ממשק המשתמש.

השרת הראשי יכול לתפעל כרגיל גם בלי ששרת הגיבוי יהיה מופעל, אך במידה ושרת הגיבוי לא קיים אין הגנה בפני נפילות של השרת הראשי והמשך פעילות תקינה.

מחשב משני - מרובה

1. פתיחת טרמינל

2. פתיחת טרמינל erl עם שם ייחודי לשרת הגיבוי וכתובת ה ip של ה localhost
erl -name simulation1@127.0.0.1 -setcookie "abc"

ברשת תקשורת (לדוגמה במעבדה)
erl -sname simulation1-setcookie abc

3. בתוך הטרמינל של ה erl ננווט לתיקיית הפרויקט
cd("C:/Users/temp/Desktop/Erlang/my2048").
עבור משתמש לינוקס אפשר לפתוח את הטרמינל ישירות בתיקיית הקבצים.

4. נקמפל את כלל הקבצים אם לא קמפלנו אותם עד כה (אפשר להפעיל ללא שרת גיבוי)

c(backup_server).
c(main_server).
c(simulation_server).
c(ui_server).
עבור גרסאת לינוקס נקמפל גם את הקובץ הנוסף
c(ui_server2).

5. נפעיל ישירות שרת סימולציה עם שם ה node שנתנו **לשרת הראשי**

simulation_server:start('main@127.0.0.1').

ברשת תקשורת (לדוגמה במעבדה)

main_server:start(main@hostname).

כאשר hostname הינו שם ה node של ה main ברשת, ניתן למצוא ע"י hostname בטרמינל.

שרת ממשק המשתמש יתעדכן במיידית בחיבור לשרת סימולציה.