פרוייקט סופי בקורס מתקדם במערכות מחשוב - Hot Potato

מגישים: ג'יי טננבאום 323639476, אריאל פרנקנטל 207020660

מבוא

הפרויקט שלנו, Hot Potato, הינו **משחק מבוזר** בין בקרים מרובים (מתקשר לעולם ה-Internet of Things), כאשר כל בקר מריץ את אותו קוד. שאבנו השראה ממשחקים כמו טמגוצ'י, שילדים קונים בחנות את אותו המוצר המריץ את אותו הקוד, ויכולים להיפגש בקבוצה ולשחק משחק קבוצתי מרובה משתתפים. השתמשנו בלוחות CC1350 של Texas Instruments, המריצים את מערכת ההפעלה TI-RTOS שלמדנו לעבוד עימם במסגרת הקורס. תכנות המשחק הכיל בתוכו אתגר תכנותי ותקשורתי, בין היתר לתכנת את לוגיקת המשחק, לעקוב אחרי הזמן וחיי המשתתפים, כיצד לגרום לכל הבקרים להריץ קוד אבל ליצור סדר מעגלי בין המשתתפים, תכנות התקשורת ועבודה עם proprietary, ועבודה עם חוטים מרובים ומנגנוני האינטראפטים.

מוטיבציה

אנו רואים שימוש אפשרי אמיתי לפרויקט, וכן קהל משתמשים לא קטן. אנו מדמיינים את שיבוץ מיקרו- הבקר בתוך מעין טמגוצ'י עם GUI יותר מתקדם מנורות בצבעים. הייצור יהיה זול יחסית, אשר יוביל לעלויות נמוכות למשחק, והורים לילדים קטנים שעדיין שאין להם פלאפון יוכלו לרכוש את המוצר בשבריר מעלות פלאפון, כך שהילדים יוכלו לשחק בגן/ בית- הספר. המשחק הוא בין כמה משתתפים, והאינטראקציה תמשוך אנשים למוצר.

תיאור המשחק

המשחק מבוסס על המשחק הידוע Hot Potato, שלפי ויקיפדיה :

"Hot potato is a party game that involves players gathering in a circle and tossing a small object such as a beanbag to each other while music plays. The player who is holding the object when the music stops is eliminated."

בוריאציה שלנו של המשחק השחקנים יעמדו במעגל, וכל אחד יתחיל עם מכסה כוללת של 10 שניות להחזיק את תפוח האדמה (אצלנו אור ירוק) עד שהוא מת. מתחילים בסבב קליברציה נגד כיוון השעון (לחיצה על 2 הכפתורים בו"ז על הבקרים לפי הסדר), ואז השחקן שהתחיל "סוגר מעגל" ומתחיל את המשחק ע"י לחיצה בו"ז. שחקן אקראי מתחיל עם האור הירוד, וכל עוד הוא לא לחץ על כפתור, הזמן הכולל שלו יורד (זהו משחק זריזות וריפלקסים). אם הוא לוחץ על הכפתור הימני, הוא מעביר את האור הירוק לשחקן לימינו, ואחרת לשמאלו. לחיצה על כפתור שלא בתורך יגרום לכך שתיפסל. כאשר לשחקן שאצלו האור הירוק עומד להיגמר הזמן, גם האור האדום דולק בנוסף לירוק, וכאשר ממש נגמר הזמן, הוא מודח ונדלקות לו 2 נורות עד המשחק הבא, ושאר השחקנים ממשיכים במעגל בלעדיו, כאשר השחקן הבא מוגרל אקראית בתור אחד משכניו של המודח. המנצח הינו השורד האחרון, וכאשר ינצח, אורותיו יהבהבו 10 פעמים. המנצח יוכל להתחיל משחק חדש ע"י לחיצה בו"ז, והוא יתחיל עם האור הירוק אצלו.

סרטוני הדגמות

סרטון ההדגמה נמצא בקישור הבא: https://photos.app.goo.gl/HnSo1tV13xbfyNwr2
(סרטון של 2 דקות המדגים את הפונקציונליות של הפרוייקט. **מומלץ לצפות באיכות מירבית!**)

קישור לקוד

הקוד נמצא ב-Git Repository בקישור הבא: https://github.com/jaytenenbaum/HotPotato.
הקובץ העיקרי הוא HotPotato.c.

לוגיקת המשחק

על- מנת שהקוד יוכל לתמוך במגוון פעולות שעלולות לקרות בזמנים שונים, כגון פקיעת שעון הזמן הנותר, קבלת הודעה משחקן אחר, או לחיצה על כפתור, אימצנו את פרדגימת התכנות **Automata-based programming**, ויותר ספציפית **FSM-based programming**, שבמסגרתה התוכנה מיוחסת כאל מכונת מצבים שנמצאת במצב מסוים.

כל בקר יכול להיות באחד מתוך 8 המצבים הבאים:

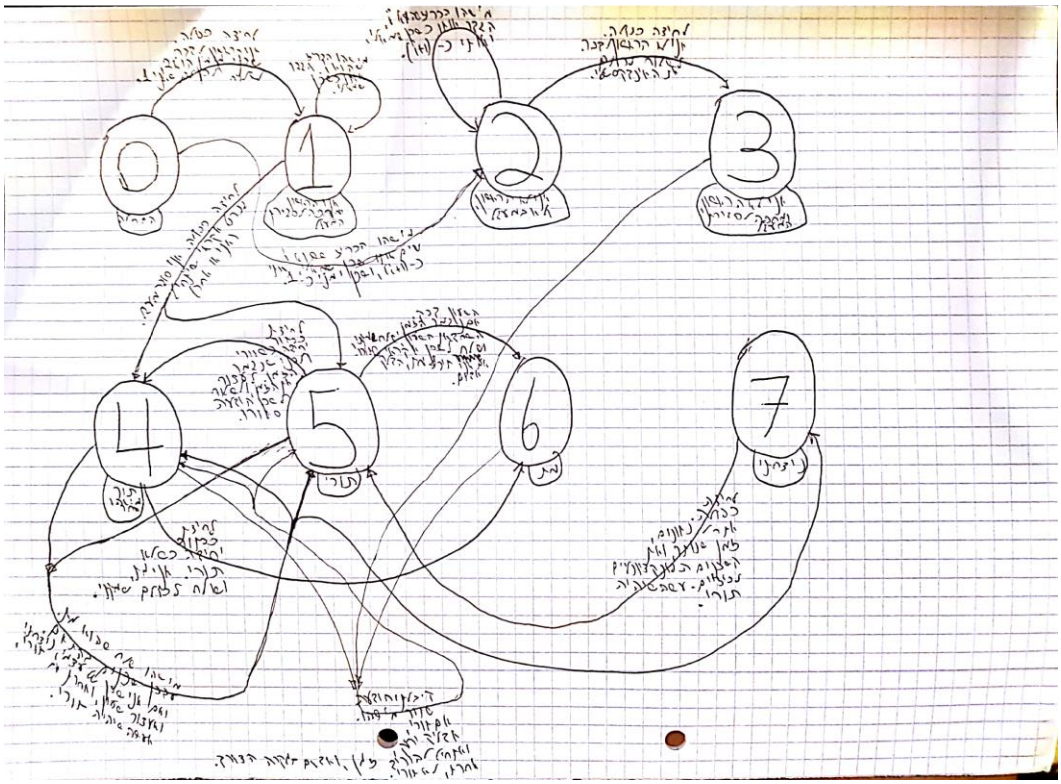
- 0 – מצב התחלתי, טרם בוצעה שום פעולה.
- 1 – אני הראשון במעגל, ואני מחכה לסגירת מעגל האתחולים.
- 2 – אני לא הראשון במעגל, וטרם הכנסתי את עצמי למעגל.
- 3 – אני לא הראשון במעגל, ואני מחכה לסגירת מעגל האתחולים.
- 4 – האור הירוק (תפוח האדמה) נמצא אצל שחקן אחר. (או לחילופין ממתין עד שיהיה תורי)
- 5 – האור הירוק (תפוח האדמה) נמצא אצלי.
- 6 – אני מת.
- 7 – ניצחתי.

ניזכר שישנן 3 סוגי פעולות שיכולות להניע את המצב של המשחק: לחיצת כפתור, קבלת הודעה, פקיעת שעון. לכן על- מנת להניע קדימה את הלוגיקה של המשחק, נרצה לדעת לכל מצב ולכל סוג קלט מה לבצע.

נתחזק גם לאורך הדרך מספר משתנים פונקציונליים:

- state – מצב המכונה שלי.
- leftFriend, rightFriend – האינדקסים האפליקטיביים במעגל של השכנים שלימיני ולשמאלי. (כלומר תוך כדי המשחק, האם השחקן שלימינך מת אז השחקן הימני שלו הופך להיות הימין האפליקטיבי שלך). השכן השמאלי מאותחל מראש ל-0 (Dummy Value), והימני ל-2.
- leftFriendPhysically, rightFriendPhysically – האינדקסים הפיסיים במעגל של השכנים שלימיני ולשמאלי. (התוצאה שלאחר שלב הקליברציה. לאתחול סבב שני של המשחק, ללא קליברציה נוספת).
- myIndex – האינדקס שלי במעגל. מאותחל מראש ל-1.
- remainingTimeHundrethSeconds – הזמן שנותר לי ביחידות של מאית השנייה. מאותחל מראש ל-1000.0, כלומר 10 שניות.

מכונת המצבים:



<u>סיכום המעברים בין המצבים:</u>

- **במצב 0 (התחלה)**:
  - אם המשתמש לחץ לחיצה משותפת על 2 הכפתורים, אנחנו הראשונים לדבר, ואנחנו אומרים לכולם שאנחנו מספר 1, וקובעים את הימין (פיזי ופונק') להיות 2.
  - אם מישהו הכריז שהוא i, אז הוא השכן השמאלי שלנו (פיזי ופונק'), ואנחנו i+1, והשכן הימני שלנו הוא 1 (פיזי ופונק'), כלומר אפקטיבית אנחנו בסוף המעגל.
- **במצב 1 (אני הראשון, ומחכה לסגירת המעגל)**:
  - אם מישהו הכריז שהוא i, אז הוא השכן השמאלי שלנו (פיזי ופונק'), כלומר אפקטיבית אנו בסוף המעגל.
  - אם המשתמש לחץ לחיצה משותפת על 2 הכפתורים, אנו סוגרים מעגל, ומגרילים שחקן אקראי שיתחיל (או שזה אני, ואז נהייה תורי, או שתור מישהו אחר, ושולחים לו שתורו)
- **במצב 2 (אני לא הראשון, ולא במעגל)**:
  - אם מישהו הכריז שהוא i, אז הוא השכן השמאלי שלנו (פיזי ופונק'), ואנחנו i+1.
  - אם המשתמש לחץ לחיצה משותפת על 2 הכפתורים, אנו אומרים לכולם את האינדקס שלנו.
- **במצב 3 (אני לא הראשון, ומחכים לסגירת מעגל)**:
  - כאשר קיבלנו הודעה שתור מישהו, אם תורנו אז נדליק אור ירוק (ונכבה את האדום אם היה דלוק), ונתחיל לספור זמן. אחרת, לא תורנו, ונעבור למצב 4 ונכבה את שני האורות אם היו דלוקים.
  - אם מישהו הכריז שהוא i, אז הוא השכן הימני החדש שלנו (פיזי ופונק'), ועוברים למצב 4 כדי לא לשנות את השכן הימני בפעם הבאה. אפקטיבית, אנחנו מוכנים למשחק.
- **במצב 4 (תור מישהו אחר)**:
  - אם מישהו שלח שהוא מת, נעדכן בהתאם את שכניי (הפונקציונליים), ואם אני שכן פונק' של עצמי אז ניצחתי ואעצור שעון, ואחרת אם תורי אז נדליק אור ירוק, ונתחיל לספור זמן.
  - אם המשתמש לחץ על כפתור, אז זו לחיצה אסורה כשלא תורי, ולכן נמות ונשלח לכולם שמתתי.
  - כאשר קיבלנו הודעה שתור מישהו, אם תורנו אז נדליק אור ירוק (ונכבה את האדום אם היה דלוק), ונתחיל לספור זמן. אחרת, לא תורנו, ונעבור למצב 4 ונכבה את שני האורות אם היו דלוקים.
- **במצב 5 (תורי)**:
  - אם המשתמש לחץ על כפתור, אז נעצור את הזמן, ונשלח הודעה לשכן שתורו.
  - אם מישהו שלח שהוא מת, נעדכן בהתאם את שכניי הפונק', ואם אני שכן פונק' של עצמי אז ניצחתי ואעצור שעון, ואחרת אם תורי אז נדליק אור ירוק, ונתחיל לספור זמן.
  - אם השעון דפק: אם נגמר הזמן, נשמיד את השעון, נשלח לכולם שמתתי ולשכן אקראי שתורו. אם אני תיכך מת, הדלק אור אדום.
- **במצב 6 (אני מת)**:
  - כאשר קיבלנו הודעה שתור מישהו, אם תורנו אז נאפס מחדש את המשחק, נכבה את זוג האורות, נדליק אור ירוק ונתחיל לספור זמן. אחרת, לא תורנו, ונעבור למצב 4 ונכבה את שני האורות אם היו דלוקים.
- **במצב 7 (ניצחתי)**:
  - אם המשתמש לחץ לחיצה משותפת על 2 הכפתורים, נאתחל מחדש את הנתונים, הזמן שנותר לי, ואת השכנים הפונקציונליים נחזיר לערך של השכנים הפיזיים. כעת נעשה שיהיה תורי.

מנגנון התקשורת

האתגר (הגדול ביותר) בפרוייקט היה לאפשר לכל בקר להיות גם קורא וגם כותב (תקשורת דו- כיוונית), וספציפית עבור scenario בו יש הרבה שחקנים. הבעייתיות הייתה שכל דוגמא סטנדרטית הכילה רק כיוון אחד או של שידור tx או של קליטה rx, וחשוב לציין כי הבקר אינו יכול להיות גם קורא rf וגם כותב בו- זמנית, ולכן הצטרכנו לתזמן בהתאם. הפרויקט שלנו מתבסס על קוד הלקוח מ-rfEasyLinkRx_CC1350_LAUNCHXL_tirtos_ccs , rfEasyLinkTx_CC1350_LAUNCHXL_tirtos_ccs.

אבטיפוס Transceiver (קורא וכותב)

שילבנו את הקוד מ-2 הפרויקטים הנ"ל, ובעזרת תזמונים סמפורים (שליחת הודעה, ולחכות לתשובה עד שנעכל את התשובה ונשלח עוד הודעה וכך הלאה...) יצרנו אבטיפוס של תקשורת בין 2 בקרים, המונעת על לחיצת כפתורים שעבדה באופן הבא:

- יצרנו שני סמפורים, כל אחד עם מונה 1, ולכל אחד החזקנו handle. האחד בשם semHandle, והשני בשם semHandle2.
- יצרנו Task אחד ל-rx בעדיפות 2, המריץ את המתודה rfEasyLinkRxFnx, שהוא הלוגיקה של ה-rx, הקורא. בתחילת הקוד נדרש אתחול של EasyLink אם לא אותחל, ולשם כך יש משתנה cnf שהוא נהייה 1 כאשר מאתחיל את ה-EasyLink, ורק מאתחלים אותו אם cnf=0. כלומר רק מאתחלים אותו פעם אחת. אם אתחלנו, נשחרר את semHandle, ואחרת נמתין עליו. בהמשך, אנו בלולאת while אינסופית, שבה:
  - מחכים על semHandle2, אז קוראים הודעה (שומרים למשתנה גלובלי recvMessageType) ופועלים בהתאם (אם קוראים 1, נחליף את מצבו של האור הירוק מדלוק לכבוי ולהיפך, ואם קוראים 2, נחליף את מצבו של האור האדום מדלוק לכבוי ולהיפך), ואז משחררים את semHandle, כדי ש-tx יוכל לרוץ.
- יצרנו Task אחד ל-tx בעדיפות 3, המריץ את המתודה rfEasyLinkTxFnx, שהוא הלוגיקה של ה-rx, הקורא. בתחילת הקוד נדרש אתחול של EasyLink אם לא אותחל, ולשם כך יש משתנה cnf שהוא נהייה 1 כאשר מאתחיל את ה-EasyLink, ורק מאתחלים אותו אם cnf=0. כלומר רק מאתחלים אותו פעם אחת. אם אתחלנו, נשחרר את semHandle, ואחרת נמתין עליו. בהמשך, אנו בלולאת while אינסופית, שבה:
  - מחכים על semHandle (עד שנייה, כדי שלא ייתקע), אז כותבים הודעה ומשדרים אותה (פעמיים, יוסבר בהמשך) ואז משחררים את semHandle2, כדי ש-rx יוכל לרוץ.
- יצרנו מאזין ללחיצות כפתורים, המקושר לפונקציה buttonCallbackFxn, שבהתאם לכפתור שעליו לוחצים, ישנה את המשתנה הגלובלי sendMessageType, שמשם נלקח הערך לשליחה ב-tx בכל איטרציה. ימין יהיה 1, ושמאל 2. הערך הזה יישאר קבוע עד שנלחץ על כפתור אחר, ומעתה ואילך המכשיר ישלח את אותו הערך המעודכן שוב ושוב בכל איטרציה.

התוצאה:

כאשר מחברים רק לוח אחד אין תקשורת (כמובן...) ואף אור לא משתנה, וכאשר מחברים 2 לוחות או יותר, הם ישלחו אחד לשני הודעות הלוך ושוב (פינג- פונג אינסופי) לכמות זמן לא מוגבלת.
אב- הטיפוס של התקשורת תרם לנו להוכחת יכולת, אך היה צורך באדפטציה שלה, שינוייה והכנסה לפרוייקט, המתואר מלרע:

השמשת ה-Transceiver לצרכי Hot Potato

כעת, נרצה להרחיב זאת כך שכל בקר רק יעבד הודעה פעם אחת (כיוון שהתקשורת תמיד בפינג- פונג, ורק נרצה להתייחס להודעה חדשה בפעם הראשונה שאני קורא אותה), **ושיעבור מידע נחוץ בהודעה לשם תגובה לפעולה בהתאם**.

תגובה יחידה לכל הודעה

- נשמור משתנה int seqNum=0, שתפקידו להעיד על האינדקס של ההודעה האחרונה שאנחנו עדים אליה, והוא מאותחל ל-0.
- כל פעם שאנו קוראים הודעה, אנו בודקים את ערכו בהודעה שנשלחה אלינו (payload[0]), ואם הוא גדול ממה שזכור אצלי שהאינדקס הכי גבוה ואז **אגיב למסר ואעבד אותו כהודעה חדשה**, ואעדכן את seqNum אצלי לאינדקס של ההודעה שעיבדתי. אחרת אתעלם לחלוטין מההודעה.
- כמובן שכשאני שולח הודעה החוצה, עליי קודם להעלות את seqNum אצלי ב-1 כי ההודעה שלי עכשיו באינדקס אחד אחרי ההודעה האחרונה שקראתי, ואני עד אליה כיוצר שלה. כך מי שיקרא את ההודעה יתייחס אליה פעם אחת בדיוק.

באופן יוריסטי כדי לשפר את האחוזים בהם ההודעה הופצה **לכל** השחקנים, עשינו שתי היוריסטיקות, כאשר הראשונה היא שבקבלת הודעה חדשה, קודם נפרסם אותו שוב כדי להעלות את הסיכוי שישמעו אותו (בעיות וקשיי התקשורת מפורטות בהמשך), וכן בשידור הודעה נבצע את השידור פעמיים במקום פעם אחת. בכל מקרה, אין מה לדאוג, וההודעה לא תעובד פעמיים, כיוון שיש את ה-seqNum שיאכוף זאת.

אופן העברת המידע בתקשורת

כעת שהצלחנו ליצור את התקשורת הרצויה בין הבקרים, נעבור למודל העברת המידע בין אחד לשני.
המידע כולו עובר ב-payload כאשר:

- ב-payload במקום ה-0 נשתמש בתור ה-seqNum, המספר הסידורי של ההודעה.
- ב-payload באינדקס ה-1 נעביר את סוג ההודעה, וב-2 עד 5 נעביר את תוכן ההודעה:
  - אם זו הודעת "תורך" (YOUR_TURN_MESSAGE_TYPE):
    - באינדקס ה-2 נעביר את האינדקס של השחקן שתורו.
  - אם זו הודעת "שלום, האינדקס שלי הוא" (HI_MY_INDEX_IS_MESSAGE_TYPE):
    - באינדקס ה-2 נעביר את האינדקס של השחקן שמדבר.
  - אם זו הודעת "אני מת" (IM_DEAD_MESSAGE_TYPE):
    - באינדקס ה-2 נעביר את האינדקס של השחקן המת.
    - באינדקס ה-3 נעביר את האינדקס של השחקן הפונקציונאלי השמאלי של השחקן המת.
    - באינדקס ה-4 נעביר את האינדקס של השחקן הפונקציונאלי הימני של השחקן המת.
    - באינדקס ה-5 נעביר את האינדקס של השחקן שתורו.

לחיצה משותפת (בו"ז) על 2 הכפתורים

רצינו לממש אופציה ללחיצה משותפת בו"ז על שני הכפתורים כדי ליצור פונקציונליות חדשה, וזאת בעזרת ה-events של כפתור יחיד.
מימשנו זאת בכך שיצרנו מאזין לכפתורים, שכאשר מזהים לחיצה על כפתור כלשהו מורצת המתודה buttonCallbackFxn. במתודה זו מחכים CPUdelay(100*8000) , ואז בודקים את מצב הכפתורים. אם שניהם לחוצים אזי יש לנו לחיצה משותפת (בו"ז), ואחרת יש לחיצה על כפתור יחיד. העניין הוא רק שכשיש לחיצה משותפת, ניכנס ללולאה פעמיים: פעם עבור הפתור שנלחץ ממש קצת לפני השני, ופעם אחת עבור הכפתור שנלחץ ממש קצת אחרי השני. לכן יש לנו משתנה שמור firstClickOfDouble, שבכל כניסה לענף שזיהינו לחיצה משותפת, יהפוך את ערכו מ-0 ל-1 ובחזרה, ורק אם הוא בערך 1 אז נבצע לוגיקה, ואחרת לא נבצע כלום. כך אפקטיבית נבצע פעולה ללחיצה המשותפת בדיוק פעם אחת לכל לחיצה משותפת (במקום פעמיים).

לכן, סה"כ יש באפשרותנו לבצע פעולה פעם אחת עבור לחיצה משותפת, וכן בעזרת switch (pinId) נוכל במקרה של לחיצה יחידה (על כפתור יחיד) לדעת על איזה כפתור לחצנו.
לכן יצרנו את המתודה doubleButtonReaction() שמורצת כאשר מזהים לחיצה משותפת, והמתודה singleButtonReaction(LEFT_BUTTON/RIGHT_BUTTON) מורצת כאשר זיהינו לחיצה על כפתור יחיד, עם הארגומנט בהתאם לכפתור שנלחץ (עשינו define ל-LEFT_BUTTON=0, RIGHT_BUTTON=1 למען הקריאות).

מעקב עם שעון

בכדי שנוכל לעקוב אחרי כל שחקן אחרי הזמן הנותר לו ,ולזהות בזמן אמת כאשר נגמר לו הזמן ,מתבקש שיהיה שעון שינהל זאת.
האופן שבו השעון עובד הוא בכך שמאתחלים אותו במתודה startGeneralTimer עם כמות הזמן הכוללת הרצויה לתחילת המשחק (זה 10 שניות אצלנו במשחק). מחברים אותו למתודה generalTimerTick שייקרא באופן מחזורי כל טיק של השעון, כפי שאתחלנו אותו. במתודה זו נשאל את עצמנו האם אנחנו במוד של הורדת זמן, כלומר האם הנורה אצלי, וזה נקבע לפי המשתנה toDeduct שבודקים שהוא אינו 0. אם זה אכן המצב בודקים אם אנחנו בסיכון (כלומר נותרו 2 Sec>) בעזרת המתודה turnRedIfInDanger. אם כן, הוא ידליק גם את חיווי האור האדום. בהמשך , נוריד מסכום הזמן הכולל הנותר, remainingTimeHundrethSeconds, את הזמן שעבר מאז הטיק הקודם כלומר timerDeltaHundrethSec ,ולבסוף, אם הזמן הנותר הוא לכל היותר 0, אזי סימן שהנורה נמצאת אצלנו, ונגמר לנו הזמן, ולכן נעבור למצב של מת, ובהקשר של השעון נמחק אותו ע"י שימוש במתודה killClock. כמובן שגם במקרה שבו אני מזהה אצלי ניצחון בעקבות קבלת הודעת מוות, נמחק את השעון.
השעון יאותחל מחדש בכל משחק עוקב דרך המתודה initGeneralGameData שקורא לאתחול השעון במתודה startGeneralTimer.

אתגרים נוספים ובעיות פתוחות

- כאשר דברים עבדו בשונה מאיך שציפינו, עבדנו עם הדיבאגר של CCS, ובהתחלה עם מנגנון של printf-ים שהדפיס מידע לאורך הדרך. אך למרבה הפלא, השיטה של printf-ים התגלתה לאורך הדרך כפוגעת בתקשורת בין הבקרים השונים, אולי כיוון שהפעולה לוקחת זמן רב, ולכן לא יכולנו עוד להסתמך על שיטה זו, כיוון שכשמפעילים אותה קורה דבר שונה מכשלא מפעילים אותה. לכן לרוב השתמשנו בדיבאגר של CCS, שלעיתים דרש מאיתנו תגובה זריזה יחסית, כיוון שהבקרים תמיד מדברים ביניהם, ולא רצינו להרוס תקשורת פינג-פונג זו.
- כפי שציינו מוקדם יותר, האתגר של התקשורת, העבודה עם rf ועם easylink היה האתגר אולי הגדול ביותר. ניסינו המוני פעמים לגרום לבקרים לתקשר, ואז שהתקשורת תעבוד, ושכל ההודעות יגיעו בדיוק פעם אחת. גילינו גם לאורך הדרך שכאשר מחברים את שני הבקרים עם אותו הכבל שמגיע עם הלוח אל המחשב, אז התקשורת עובדת פלאים, וכאשר מחברים אחד מהם לשקע חשמלי עם כבל אחד, ואת השני למחשב, אז התקשורת יכולה לא לעבוד ברמה שהודעה אחת מתוך חמש לא מגיעה ליעד, והמשחק נהרס (לדוגמא, אם שחקן מוסר את האור הירוק אל השכן שלו, והשכן לא מקבל אותו, אז האור הירוק "נאבד", ולא תור אף אחד יותר, ונדרש להתחיל מהתחלה.)
לכן המסקנה הראשונה שלנו היא לחבר את כל הבקרים באותו אופן אל המחשב.
כאשר זהו המצב, אז התוכנה המוקדמת יותר שעשינו, שבמסגרתה רק שני בקרים מתקשרים ביניהם, עובדת מעולה, ואולי רק אחת מתוך כ-50 הודעות נאבדת, שזה בהחלט מספיק לכמה משחקים מלאים עד שמרגישים בכך. כאמור, אם אחד מחובר לחשמל, קשה לנהל רצף משחקים ללא שגיאה.
לעומת זאת, בתוכנה המתקדמת יותר, שבמסגרתה מספר השחקנים הוא מספר גדול מ-2, התקשורת עובדת פחות טוב (זה לא מפתיע, כיוון שצריך להבטיח שיותר הודעות יגיעו אל היעד, ויש יותר הודעות ש-"נזרקות לאוויר". "יותר קשה לנהל שיחה בין 100 אנשים במעגל, מאשר אם היו רק 2".). זה עובד ברמה שלעיתים ההודעה הראשונה לא מצליחה לעבוד (נאמר כ-20% מהפעמים), ואם ההודעה הראשונה הצליחה, אז ממש לרוב מצליחים לעבור דרך מרבית המשחק, ועד סוף הסבב הראשון יש עוד כ-20% סיכוי שהתקשורת תיפול. לכן בעיית התקשורת של 100% היא עדיין **בעיה פתוחה** מבחינתנו למקרה בו יש שחקנים מרובים.
<u>השקענו שעות רבות במחקר על כיצד לתקן את המצב, והבאנו לשיפור משמעותי במצב לעומת ההתחלה:</u>
<u>*</u> מימשנו פונקציית hash על ההודעה, שנשלח כקוד למציאת שגיאות בהודעה, כדי שכאשר נקבל הודעה, נוכל לדעת האם אין בו שגיאות. מאוחר יותר הורדנו אותו כיוון שהוא נמצא פחות אפקטיבי.
<u>*</u> מודל התקשורת בוסס על כך ששולחים את ההודעות שוב ושוב בפינג- פונג, ולכן אם ההודעה לא הגיעה בפעם הראשונה טוב, אז יש עוד נסיונות.
<u>*</u> הפעלנו שתי היוריסטיקות והן: 1. הפצה מחדש של הודעה שקיבלתי לפני שאני מעבד אותה. 2. הפצת הודעה פעמיים במקום פעם אחת. שתי שיטות אלו נמצאו כאפקטיביות, ואכן שיפרו משמעותית את התקשורת, ממצב שרק בערך אחת ל-3 הודעות מגיעה, למצב שבממוצע הרוב המוחץ מגיע, ולכן כך זה מומש בפרוייקט.
- האתגר שצריך שיהיה קוד זהה ואחיד לאורך הבקרים דרש מאיתנו לחשוב מחוץ לקופסא. קיבלנו אתגרים כמו כיצד לקבוע מי הראשון במעגל, כיצד שכל אדם יידע מי נמצא בצדדיו, וכיצד כל שחקן ינהל לעצמו זיכרון ייחודי לו, למרות שלכולם אותו הקוד. נעזרנו בסוף ב- FSM-based programming, שצויין בהתחלה, בכדי לענות על אתגר זה, ועל האתגר שגורמים רבים יכולים לשנות מצב של בקר (לדוג' קבלת הודעה, לחיצה על כפתור, או פקיעת שעון) .

מילות סיכום

הפרוייקט היווה אתגר מרתק ומאתגר בעבודה בתכנות מערכות משובצות, ובהתעסקות עם התקשורת בין רכיבים כאלו. נוכחנו לאתגרים בהפצת קוד אחיד, בבניית הלוגיקה של המשחק, בעבודה עם החומרה, ובניהול החוטים ובסביבה מונחית אינטראפטים. אנו גאים במוצר (משחק) Hot Potato שיצרנו, באתגרים הרבים שהצלחנו לעמוד בהם, ומרוצים מהידע הנוסף שרכשנו במסגרת ה-"כניסה בעובי הקורה" שנדרש מבניית הפרוייקט.