דוח מסכם פרוייקט בקורס "תכנות פונקציונאלי בסביבת JVM
CSS SPRITE

מוגש ע"י:

דן וינשטין	038103495
דרור פדידה	039062658
דניס סיבורקש	319446746

תאריך הגשה: 25/02/13

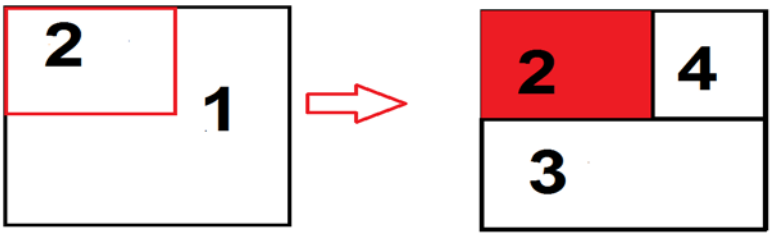
תיאור הפרויקט

בפרויקט מימשנו css sprite genetator .

הבעיה: ריבוי פניות של הדפדפן לשרת כדי להציג תמונות בעת העלאת דף.
הדפדפן פונה לשרת עם בקשה לכל תמונה בנפרד ללא קשר לגודל התמונה.
לכן יכול להווצר מצב של בזבוז רוחב פס על המון בקשות קטנות.
יצירת תמונה גדולה מכל התמונות הקטנות.
יצרית קובץ בפורמט css שימפה את מיקומה של כל תמונה קטנה בתמונה הגדולה לפי קוארדינטות וגודל התמונה.

הגישה לפתרון: נאחד את הפניות של הדפדפן לפניה אחת עבור תמונה גדולה שתכיל את כל התמונת הקטנות שדרש הדפדפן ע"י איחוד התמונות לתמונה אחת גדולה שאותה השרת יספק לדפדפן בעת בקשת האובייקטים. נשאף ליצור תמונה שבה כמה שפחות חללים ריקים ,שרוב שטחה בנוי מהתמונות הקטנות.
סידור התמונות הקטנות בצורה יעילה תתרום לתקורה מינמלית בהעברת הנתונים מהשרת לדפדן ומכאן יעילות מקסימלית של רוחב הפס.
בעיית סידור תמונות שגודלם משתנה על יריעה זהה לבעית הbin packing הידועה, והיא מוגדרת כבעיה NP-HARD לכן מימוש פתרון יעיל הוא לא ריאלי, לכן נממש **פתרון חמדני**
כרגע הפורמט תמונה היחידי שנתמך הוא PNG כל תמונה אחרת לא תכנס לאלגוריתם .

הפתרון: נגדיר חלל ראשוני בגודל שרירותי מקסימלי כאשר רוחבו כרוחב התמונה המקסימלית ואורכו כאורך סכום אורכי התמונות.
נמיין את התמונות לפי רוחבן ונתחיל למקם אותן בחלל הראשוני מהגדולה לקטנה באופן הבא:
ברגע שנבחר חלל התמונה ממוקמת בפינה השמאלית העליונה שלו,אז מועבר קו אורך אחד אחרי התמונה שמפצל את החלל המקורי לשניים ואז קו אנכי יורד מהצלע הימנית של התמונה מהצלע העליונה של החלל המקורי עד לקו אורך החדש , להמחשה:



1=the original space (including the part that the picture is going to take
2= the picture
3,4= the new spaces

<u>האלגוריתם</u>:

נתחזק אוסף מפות שמורכב מחללים פנויים SpacList, חלל פנוי ייוצג ע"י מפה עם המפתחות גודל החלל והקואורדינטות.

נחזיק אוסף מפות של תמונות ממויין מהרחבה לצרה sortedPics, כל תמונה תיוצג ע"י מפה עם הנתונים שלה.

נספק וקטור ריק עבור התמונות שמוקמו LocatedPictures.

1. לכל תמונה בוקטור sortedPics

1.0 אם וקטור sortedPics ריק החזר את LocatedPictures (תנאי עצירה לרקורסיה)

1.1 מצא חלל טוב ביותר

1.2 במקם את התמונה בחלל (ייצר ערך תמונה חדש עם קואורדינטות חדשות) וייצר חללים חדשים.

1.3 עדכן את רשימת החללים updateSpaceList

1.3.1 מחק את החלל הנ"ל מהרשימה

1.3.2 הכנס במקומו שני חללים חדשים

1.4 הוסף את התמונה ה"חדשה" לוקטור LocatedPictures

1.5 חזור ל1 עם updateSpaceList שאר sortedPics וLocatedPictures

2. עם סיום הריצה צור תמונה צור תמונה חדשה ע"פ :

2.1 רוחב = רוחב התמונה הגדולה ביותר

2.2 גובה = התמונה עם ה Y + גובה תמונה הגבוה ביותר מ sortedPics

האלגוריתם הנ"ל ממומש בפונקציה **getBounds** בקובץ manageImages שורה 108

המימוש הוא רקורסיבי ע"י חזרה עם recur ופרמטרים שונים עם תנאי עצירה על גודל הקלט המתקצר -sortedPics

האלגוריתם הנ"ל ממומש באופן חמדני ,כלומר ברגע שתמונה מוקמה ב SPRITE היא לא תזוז ממיקום זה . החלל הטוב ביותר נבחר כך:

נבחר את החלל הצר ביותר שהתמונה יכולה להכנס בו ברגע הנתון מבלי להתחשב בשאר הקלט(גובה ,או תמונה עם רוחב זהה אך גבוהה יותר שתוכל לאכלס חלק גדול יותר מהחלל)

פתרון זה ממומש בפונקציה **GetBestSpace** בקובץ manageImages שורה 52.

זוהי פונקציה רקורסיבית עם חזרה recur וקלט מתקצר (נשים לב שאין תנאי עצירה כי תמיד נגיע לחלל גדול דיו שיוכל לאכלס את התמונה כי הגדרנו את גודל החלל הראשוני באופן מקסימלי)

בכל איטרציה נבצע do בדיקה אם התמונה מתאימה בחלל הנוכחי.

בשלב 1.2 נקרא לפונקציה **PlacePic** בקובץ manageImages שורה 65.

פונקציה זו מקבלת את התמונה הנוכחית ואת החלל הטוב ביותר שנבחר עבורה ומחזירה שני חללים חדשים ואת התמונה עם הקואורדינטות החדשות.

ביצענו binding ל NewPic, SpaceB ,SpaceA עם הערכים החדשים שלהם לפי חלוקת החלל המקורי. SpaceA , SpaceB יכולים להיות גם nil במידה והגענו לגבולות החלל הראשוני.

בשלב 1.3 נקרא לפונקציה **updateList** בקובץ manageImages שורה 92.

נבצע filter וניקח את כל החללים שהקואורדינטות שלהם הם לא של החלל שהשתמשנו ב1.2 כלומר נמחוק אותו בהוקטור.

נוסיף את החללים החדשים SpaceA, SpaceB ונבצע פילטר לערכים שהם nil.

שלבים 1.4,1.5 מבוצעים בקריאה החוזרת לפונקציה **getBounds** ע"י פקודת recur בשורה 124
הוספת התמונה החדשה לוקטור LocatedPictures תבוצע ע"י הפקודה conj והקטנת הקלט תתבצע ע"י פקודת (rest SortedPics) שלוקחת רק את האיברים מהשני עד האחרון.

<u>פונקציות חשובות נוספות:</u>

הפונקציה העקרית בקוד, המנוע של האלגוריתם היא כאמור הפונקציה **getBounds** אך קיימות גם הפונקציות שיוצרות את הפלט:

הפונקציה **writeToFile** הנמצאת בקובץ writeCSS שורה 33 מקבלת את וקטור LocatedPictures ומדפיסה אותו בשני פורמטים לשני קבצים שונים :

א. פורמט css, יכתב לקובץ sprite.css שעבור כל תמונה תרשם שורת הקוד
{#cssClassName width: x ;height: y ;background-position: u v ;}

כאשר x,y הם מימדי התמונה ו u,v הן הקואורדינטות שלה בsprite.

ב. פורמט HTML יכתב לקובץ sprite.html שעבור כל תמונה תרשם שורת הקוד
< img src ="imageName.png" style = "width: 128px
"height: 128px; position: absolute; top: 0; left: 0 >
(קובץ ה HTML הוא יותר להמחשה ולא ממש מחויב בפתרון של css-sprite)

הכתיבה תתבצע ע"י הפעלת doseq על כל התמונות בLocatedPictures .

הפונקציה **combine-images** הנמצאת בקובץ manageImages שורה 39
מקבלת את הוקטור LocatedPictures וקובץ פלט ויוצרת לתוכו את התמונה הגדולה.
יוצרים תמונה ריקה במימדים: רוחב התמונה הרחבה ביותר אורך: התמונה הממוקמת בקואורדינטה הגבוה ביותר+ גובה התמונה הנ"ל.
ע"י פונקציות ספריה של java המטפלות באיחוד תמונות נבצע doseq על כל התמונות ב LocatedPictures ונצייר אותן בתמונה הריקה שיצרנו לפי הקואורדינטות שלהן.

הפונקציה שמפעילה את כל הפרויקט היא למעשה **main** בקובץ core שורה 19, היא מקבלת ספרית תמונות כארגומנט ומעבירה אותו לפונקציה **generate-css-sprite** באותו קובץ בשורה 7
פונקציה זו למעשה יוצרת את sortedPics ע"י קריאת לפונקצית העזר **collecte-imgs** וקוראת ל**getBounds** ולאחר מכן ל**combine-images, writeToFile** .

<u>חלופות אפשריות לשיטת הפתרון:</u>

בעיית בחירת הפתרון האופטימלי נעוצה בבחירת החלל הריק למיקום התמונה הבאה כמו שנזכר לעיל.
אנו בחרנו במימוש חמדני אבל ישנם פתרונות נוספים:

בתחילה מימשנו פתרון שיצר תמונה אחת גדולה באופן סיראלי כלומר מהגדולה לקטנה אחת אחרי השניה (או אחת מתחת לשניה). פתרון זה הוא **פתרון נאיבי** שיוצר עימו תקורה עצומה בשטחים ריקים, מכיוון שמתחת לכל תמונה שמתחתיה יהיו תמונות קטנות ממנה יווצר שטח ריק כפונקציה של הפרש הגדלים ביניהן. שטח זה הוא אומנם "ריק" אבל הוא מהווה תמונה (לבנה/שחורה) שתופסת משקל בקובץ הסופי.

פתרון נוסף היה יצירת כל האפשרויות למיקום התמונות הנתונות בשטח נתון ובחירת האפשרות עם תקורת השטח הריק הקטנה ביותר אך פתרון זה הוא אקספוננציאלי ולא יעיל (הזכרנו בתחילה כי בעיה זו הי NP קשה) .

אנו מניחים שניתן למצוא שיטות חמדניות נוספות שינתו אולי ביצועים טובים מהשיטה שלנו אך בחרנו את שיטה זו כי היא די ברורה להבנה ודיי מספקת מבחינת ביצועים כפי שנתאר בחלק הבדיקות בהמשך.

השיטות הרלוונטיות הקשורות לתכנות פונקציונאלי:

- **רקורסיה**: מימשנו פונקציות באופן רקורסיבי לדוגמא (**getBounds** שורה 124, **GetBestSpace** שורה 52 בקובץ manageImages).
כבר הסברנו על פונקציות אלה אך נציין שהן פעלו על וקטור שבכל איטרציה הוא הולך וקטן.

- ביצענו **binding** בעזרת פקודת let המאפשרת הגדרת ערכים עבור קטע קוד ספציפי.
השתמשנו ב **doseq** המאפשר להפעיל את אותו גוף קוד עם אותו binding מספר פעמים כ – list comprehension . למשל בפונקציה **writeToFile** בקובץ writeCss בשורה 43 .
כמו כן השתמשנו גם בDO בGetBestSpace בmanageImages בשורה 55.

- השתמשנו בעקרון **הimmutable** בclojure . למשל ב **getBounds** שורה 124 השתמשנו בפקודה **conj** היוצרת אוסף חדש מאוסף הישן ומפריט חדש או ע"י הפקודה assoc המחזירה מפה ע"י צירוף מפתח/ערך חדש למפה ישנה למשל ב**PlacePic** שורה 83 באותו קובץ הנ"ל.
השתמשנו **filter** שמחזיר subsequence חדש עבור sequence מקורי.
השתמשנו ב**map** הלוקחת אוסף מקור ומפעילה על כל איבר באוסף פונקציה המסופקת איתה ומחזירה את האוסף החדש.

- השתמשנו **בפונקציות אנונימיות** – מקל על מימוש פונקציה בתוך פונקציה וניתן לממש אותן בשורה אחת למשל בקובץ manageImages שורה 102
- השתמשנו ב**namespaces** כדי להקל על קריאות הקוד, כמו כן מאפשר ליצור ב**namespaces** שונים משתנים עם שמות זהים והם יזוהו רק ב **namespace** בו הם נמצאים.
- יבאנו פונקציות ספריה מJAVA לשם יצירת התמונה הסופית.

הרחבות עתידיות אפשריות לפרוייקט:

- מציאת אלגוריתם יעיל יותר למיקום התמונות מבחינת זמן ותקורת שטח התמונה.
- החלפת מבני הנתונים שהשתמשנו במבני נתונים מתאימים יותר כגון struct .
- יעול הקוד בשימוש גדול יותר בlaziness .
- שימוש בagents כדי ליצור כמה תמונות במקביל, נניח רק תמונות מאותו גודל ואז לאחד אותם לתמונה הסופית.
- הרחבת האלגוריתם לעוד סוגי קבצים (אנחנו התמקדנו בpng)

<u>הבדיקות שנעשו לתוכנה:</u>

הבדיקות שביצענו היו לפי סוגי התמונות שבחרנו לספק לתוכנה.

בתחילה בדקנו תמונות באופן אחיד וקיבלנו טור של תמונות כאשן ניתן לראות שהתקורה היא אפס

(התמונה אופקית לצורך חסכון מקום בדו"ח)

לאחר מכן השתמשנו במגוון רחב של גדלים כדי לראות איך האלגוריתם מתנהג למשל:

ניתן לראות את חלק מהתקורה בשטח באיזורים המסומנים באדום.

אם זאת , מדדנו את משקלי התמונות בקילו בתים ונוכחנו לראות שאפילו עם תקורת השטח , התמונות הגדולות שוקלות פחות מסכום התמונות הקטנות, למשל עבור התמונה האחרונה , משקלה 87kb לעומת סכום משקלי התמונות בה שהוא 105kb ואפילו אם מוסיפים את קובץ הsprite.css משקלו זניח (כ 3kb) לכן ניתן להראות יעילות של $1-\frac{81}{105}=0.22$ כלומר חסכנו כ22% רק בגודל המידע, בהנחה שהמידע היה עובר במספר גדול של חבילות ברשת , בכל חבילה התקורה גבוהה, לכן חסכנו אחוזים נכבדים בתעבורת רשת.

ניתן לראות (למשל ע"י הרצת ספריית test5) שככל שמספר התמונות עולה היעילות עולה גם כן.

<u>הוראות הרצת הפרוייקט:</u>

הפרוייקט יוצא לקובץ CssSprite.zip אזי ניתן ליבא אותו לאקליפס

בפרויקט נמצאים ארבעה קבצי קוד core.clj , dmanageImages.clj , writeCss.clj , strTools.clj

וחמישה ספריות עם תמונות בספריה ראשית images : test1 test2….test5

מהקובץ הראשי (core) נקרא ל repl וכשהוא עולה שורת ההפעלה היא :

("#images//test" -main) כאשר את # ניתן להחליף במספרים 1-5 .

(ניתן להוסיף עוד ספריות עם תמונות משלכם באותו המיקום ולקרוא להם גם כן)

הפלטים ימצאו בספרייה הראשית של הפרוייקט והם : "sprite.css", "sprite.html","sprite.png"

<u>שימו לב</u>: עבור כל הפעלה של התוכנה עם קריאת פונקצית main עם ספריה שונה, קבצי הפלט ידרסו לקבצים החדשים.