Bubble Trouble

פרוייקט גמר במחשבים

תעודת זהות: 209424803

פיתוח: יובל בקירוב

מנחה: יהודה אור

מכללה: הכפר הירוק

תאריך: 24.5.18

Bubble Trouble

תוכן עניינים-

מחלקות –

מבוא

המשחק פותח בסביבת העבודה Visual Studio 2017 בשפת התכנות C# על ידי שימוש בספריית 4.0 XNA, תוספת חינמית ל-Visual Studio הכוללת ממשק עבודה נוח ואוסף נרחב של מחלקות וכלים מובנים המיועדים למפתחי משחקים.

MonoGame מאפשר לנו להשתמש ב- XNA בשפת התכנות #C.

מטרת הפרוייקט

המטרה העיקרית של הפרויקט הייתה רכישת ניסיון בעבודה נרחבת בעלת ממדים גדולים, הדורשות שימוש שוטף בכישורים הנחוצים בתחום התכנות ובתכנות מונחה עצמים בפרט ומכסה נושאים ועקרונות תכנותיים רבים.

מטרות והשלכות משמעותיות נוספות של ביצוע מטלת הפרויקט היו הרחבת הידע והבקיאות האישית שלי בשפת C# ופיתוח הבנה עמוקה יותר של תכנות מונחה עצמים באופן כללי, שכן הפרויקט כולו הורכב ונבנה משלביו הראשונים ממחלקות ועצמים בעלי תפקידים שונים המתקשרים ומגיבים זה לזה באופנים רבים, ובכך דואגים לתפקודו המלא של המשחק ולרצף ההתרחשויות הלוגי שבו.

פיתוח המשחק דרש תכנון מודולרי ברמה גבוהה ואבחנה מוקדמת בין חלקים שונים בתהליך כתיבת הקוד, החל מבניית מחלקות הבסיס הכלליות האחראיות לציור ולאנימציה ועד לבניית המחלקות הקטנות המנהלות את התוכן הפיזי במשחק.

כמו כן, יש לציין שהעבודה האינטנסיבית על הפרויקט עזרה לי לפתח חוש לזיהוי ואבחנה של הטעויות שלי ביתר קלות וללמוד להימנע מהן בעתיד.

בכך תרם הפרויקט לשיפור קצב העבודה שלי ולצמצום זמן החשיבה הדרוש לי על מנת להגיע לתוצאות ולתוצרים ממשיים.

מבוא לתכנות

#C - היא שפת תכנות מרובת פרדיגמות, שעיקרה מונחית עצמים, שפותחה על ידי צוות של חברת מיקרוסופט, כחלק מיוזמת NET. שלה. השפה הוצגה לראשונה בשנת 2000. בשנים 2005 – 2006 תוקננה השפה על ידי הארגונים ECMA וארגון התקינה הבינלאומי. התחביר של #C מבוסס בעיקר על זה של שפת התכנות ++C ו Java, אך הקונספט התכנותי שלה מושפע מ VB. באפריל 2010 פורסמה במקביל להצגת גרסה 4 של פלטפורמת NET.

מקור השם #C הוא פרפראזה על כך, ומשמעו במוזיקה "הוספה של חצי טון לתו " C כלומר התו" דו דיאז".

שפת #C היא שפה מרובת-פרידיגמות, שהיא בעיקרה מונחית עצמים, אך ביחד עם זאת היא גם מונחת אירועים, ובעלת תמיכה בתכנות פונקציונלי. השפה בעלת טיפוסיות חזקה. מערכת הטיפוסים בשפה היא סטטית – אך תומכת (החל מגרסה 4) גם בטיפוסיות דינמית – ובטוחה, אם כי מאפשרת גישה ישירה לזיכרון בעזרת מצביעים (בדומה לשפת ++C) באיזורים המכריזים על כך המפורש (Unsafe). שפת #C נשענת על NET Framework., הכוללת ספרייה סטנדרטית גדולה ועניפה, וסביבת זמן ריצה מתוחכמת.

שפה זו תומכת בהימשכות (Serialization) שבה נשמר רצף בין הזיכרון לדיסק, וניתן לשמור אובייקט לקובץ ולשחזר אותו מקובץ, במקביליות (Concurrency) שבה ניתן לבצע פעילות אסינכרונית של עצמים שונים. וכן מאפשרת התבוננות פנימה (Reflection) , כלומר ניתן לחקור בזמן ריצה מאפיינים שונים הקשורים לשפה עצמה, למשל לבדוק מהו הטפוס המדויק של אובייקט מסויים.

תכנות מונחה עצמים –

תכנות מונחה עצמים או לעיתים תכנות מוכוון-עצמים, הא פרדיגמת תכנות המשתמשת ב"עצמים" (אובייקטים) לשם תכנון תוכניות מחשב ויישומים. הפרדיגמה מספקת למתכנת מספר כלי הפשטה וטכניקות ובהן הורשה, מודולריות, פולימורפיזם וכימוס.

טכניקות אלו שימשו בפיתוח תוכנה החל מתחילת שנות ה – 80 ואילך, אך השימוש בפרדיגמה בשלמותה החל רק בשנות ה 90. רוב שפות התכנות המודרניות תומכות בתכנות מונחה-עצמים. תכנות מונחה-עצמים היווה מהפכה בכתיבת תוכנה, ודרש מהמתכנתים התייחסות אחרת לפתרון בעיות ורכישת הרגלי תכנות חדשים. תכנות מונחה-עצמים הוא חלק מתפיסת פיתוח מונחית-עצמים, הכוללת גם ניתוח מערכות מונחה-עצמים (OOA), עיצוב מונחה עצמים (OOD) ובמידה חלקית גם בסיסי נתונים מונחי עצמים (Object-Oriented Databases).

תבנית המחשבה של תכנות מונחה-עצמים צמחה מהתכנות הפרוצדורלי והמוגולרי שבוסס של שגרות ופונקציות שפעלו על מבני נתונים חופשיים. שיטה זויצרה קושי רב, מכיוון שכל שינוי בהגדרת משתנים בתוכנית חייב שינוי בפונקציות שפעלו בכל מרחב התוכנה. ככל שהתוכנה הייתה גדולה ומורכבת, הקושי הלך והתעצם, דבר שגרר תחזוקה רבה ואיטיות ומורכבות בפיתוח ככל שהתקדם, וכך, בניגוד למחירה למחירי החומרה שכל הזמן ירדו, מחירי התוכנות דווקא האמירו. לפיכך היה צורך לחפש דרך חדשה שתפשט ותקל על עבודת התכנות.

מרכיב בסיסי בתכנות מונחה עצמים הוא המחלקה (Class). מחלקה היא מבנה מופשט בעל תכונות (Properties) המגדירות ומאפיינות את המחלקה כלפי חוץ, ופעולות (Methods), שהן פונקציות ייחודיות למחלקה. בחלק משפות התכנות קיימים גם אירועים (Events), שהם שגרות השייכות למחלקה ומוזנקות בהקשרים שונים, למשל בתחילתו או בסיומו של הליך או כתגובה לקלט מהמשתמש.

קצת מושגים..

מושגים מעולם התכנות שבהם נעזרתי:

תכנות מונחה עצמים-

תכנות מונחה עצמים הוא למעשה חיקוי חשיבת האדם. כאשר אנו מסתכלים על העולם אנו מסווגים באופן טבעי את כל העצמים והמושגים לפי קטגוריות. לדוגמה העולם מחולק ל-חי ,צומח ודומם . את החי אנו מסווגים ל-יונקים, עופות ודגים. את העופות אנו מסווגים שוב לקטגוריות נוספות. מכאן שכל עצם או מושג עליו אנו חושבים, נמצא במספר רב של קטגוריות. מיון זה לפי קטגוריות \ תת מושגים עוזר לנו לשלוט ולסדר את המידע העצום אתו אנו מתמודדים .לדוגמה -אם אנו רוצים לתאר למשהו דגם מסוים של מכונית -אנו צריכים לתאר רק את התכונות המייחדות את אותו הדגם אבל לא צריך להוסיף כי למכונית יש ארבעה גלגלים, מנוע, תא מטען, הגה וכו'. כיוון שאנו מיינו את העצם -במקרה זה -מכונית ,לסוג מסוים ,חסכנו מלהזכיר את כל התכונות הטריוויאליות של המכונית.

יתרונות פיתוח מונחה עצמים:

- מידול נכון יותר של העולם האמיתי.
- מעבר טבעי יותר בין שלבי הפיתוח:
  - ניתוח.(OBJECT ORIENTED ANALYSIS)
  - תיכנון.(OBJECT ORIENTED DESIGN)
  - מימוש\ תכנות.(OBJECT ORIENTED DESIGN)
- יכולת טובה יותר בשימוש חוזר בתוכנה.
- תמיכה טובה יותר בתוספות ושינויים לתוכנה קיימת.

תכנות מונחה עצמים בנוי על שלושת מושגים אלו:

הורשה -

אובייקטים המשתפים חלקים מהממשק שלהם, משתפים לעיתים קרובות גם התנהגות. ירושה מאפשרת לממש את ההתנהגות המשותפת הזאת פעם אחת. רוב השפות מונחות העצמים משיגות את היכולת הזאת דרך מחלקות, שכל האובייקטים הנוצרים דרכן משתפים התנהגות – ודרך מנגנון שנקרא ירושה, המאפשר למחלקה אחת להרחיב מחלקה אחרת – להוסיף לה מימוש והתנהגות דרך מימושים של מתודות חדשות, הוספה של משתנים, ובמקרה הצורך ביצוע של דריסה של מתודות שהוגדרו במחלקה ממה יורשים.

ניתן להגיד מחלקה חדשה על בסיס מחלקה קיימת. למחלקה החדשה ישנן כל התכונות והפעולות שירשה מהמחלקה שעל פיה הוגדרה, ובנוסף ניתן להגדיר פעולות נוספות במחלקה החדשה, או לשנות פעולות שירשה. המחלקה המורישה קרויה מחלקת בסיס (base class), מחלקת אב (parent) או מחלקת-על (Superclass). המחלקה היורשת קרויה מחלקה נגזרת (derived class), בן (child) או תת-מחלקה (subclass).

דוגמא: על בסיס המחלקה "רכב" שהוא רכב שנע על גלגלים בעזרת מנוע, אפשר להגדיר את המחלקות "אופנוע", "משאית", ו "מכונית", כאשר בכל אחת מהן יהיה שינוי בצורת הרכב, במנוע ובגלגלים. על בסיס

המחלקה "מכונית" ניתן להגדיר מחלקה חדשה "מונית", ובה תכונות ופעולות נוספות: כמו "כובע" המונית הפעלת והפסקת מונה.

הדוגמא המובהקת ביותר לרעיון ההורשה מופיעה במיון עולם הטבע. בשיטת המיון ההיררכי הנהוגה בתחום זה, מוגדרות מחלקות ברמות אחדות. בכל אחת מהרמות נקבעות תכונות מסוימות, וכל רמה יורשת את התכונות של הרמות שמעלייה. כאשר אנו פוגשים עצם מסוים בעולם החי, למשל כלבה ששמה "לאסי", אנו לומדים כל תכונותיו על פי המחלקה שאליה הוא משתייך.

ישנן שפות שבהן ניתן לרשת בו זמנית מכמה מחלקות בסיס, וליצור מחלקה שממזגת את התכונות של כמה מחלקות ;ירושה כזאת עלולה ליצור בעיות במקרה של "ירושת יהלום", כלומר מקרה בו שתיים מהמחלקות הבסיס יורשות מאותה מחלקה. בשל כך חלק מהשפות מגבילות את המנגנון לירושה ממחלקת בסיס אחת, ומימוש מספר משקים.

פולימורפיזם(רב־צורתיות/רב־טיפוסיות)-

היא היכולת של אובייקט שנוצר מטיפוס מסויים להיות בו זמנית אובייקט מטיפוסים אחרים בעץ התורשה שלו, כך שניתן להריץ עליו מתודת של מחלקות שונות. באמצעות מנגנון הפולימורפיזם ניתן להתייחס למספר אובייקטים מטיפוסים שונים הנכללים בעץ תורשה בצורה אחידה דרך מאפיין בסיסי המשותף ביניהם. כאשר האובייקטים הם מטיפוסים שונים, ניתן להסתכל על המחלקה המשותפת לכולם 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) 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 בעלות 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(casting-down).

כללי בסיס בפולימורפיזם:

- בעבור מתודות הקיימות הן במחלקת הבסיס והן במחלקה היורשת (מתודות דורסות - Overriding Methods) תמיד תרוץ הפונקציה הדורסת הנמוכה ביותר בעץ התורשה (של האובייקט הנגזר - החזק יותר), העדכנית ביותר.

<u>הערה:</u> מדובר על מתודות דורסות אשר החתימות שלהן (כותרות הפונקציה) זהות לאלה של המתודות הנדרסות.

- כאשר הפניה חלשה מצביעה על אובייקט חזק יותר, ניתן להריץ דרכה רק פונקציות אשר מופיעות הן במחלקת הבסיס והן במחלקה היורשת או במחלקת הבסיס בלבד. כלומר, לא ניתן להריץ פונקציות השייכות לאובייקט החזק ואינן דורסות (שמופיעות רק בו).

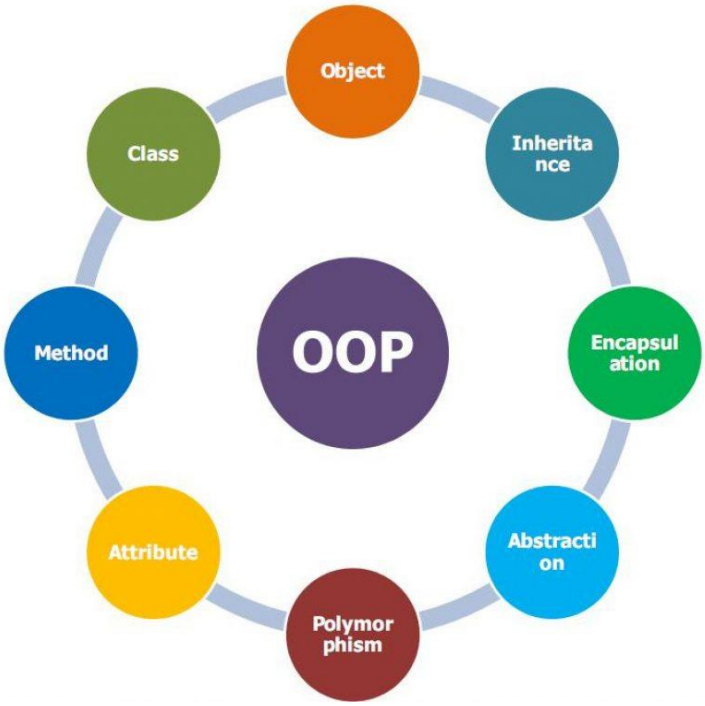
- כדי להריץ מתודות כאלה יש לבצע המרה למטה (casting-down) להפניה, כך שתהיה מאותו הטיפוס של האובייקט שבמחלקתו קיימת אותה פונקציה, ובכך להשיב לה הרשאות ולהריץ את
- הפונקציה

עיקרון הכימוס (Encapsulation) -

המימוש הפנימי של אובייקט הוא באופן כללי מוסתר מחלקי קוד החיצוניים להגדרת האובייקט. רק המתודות של האובייקט עצמו (או לעיתים, של אובייקטים אחרים השייכים לאותה מחלקה) רשאיות לבצע פעולות על השדות שלו.

יצירת האובייקט כמודול סגור שכל תוכנו מוסתר מפני המשתמש, מאפשרת בקרה טובה על כל הפעילות עם האובייקט. כל הנתונים הפנימיים (כמו גם פעולות פנימיות) המשמשים את העצם, אינם ידועים מחוץ לעצם (כלומר למתכנת המשתמש בו) ולכן מפתח האובייקט יכול לשנותם בחופשיות מבלי שיהיה צורך לשנות את התוכניות אשר משתמשות באובייקט. הדבר גם מאפשר להחליף בקלות ופשטות מנועי תכנות אשר פועלים מבחינה פנימית באופן שונה לגמרי, כל זמן שהם נותנים כלפי חוץ את אותם שיטות ומאפיינים.

הכימוס מקל על יצירת תוכנית מודולרית, פשוטה להבנה וניווט, וקלה לתחזוקה.



פעולה –

בתכנות נקראת פרוצדורה או שגרה, והיא רצף של פקודות שמאוגדות יחד, וזאת במטרה לבצע מטלה מסוימת או מימוש של אלגוריתם. הפעולה יכולה להחזיר או לא להחזיר ערך, במידה ופעולה אינה מחזירה ערך היא כנראה משמשת לשם שינוי מצב כלשהו, הדפסות למסך, שינוי של אובייקט או לגרום לתוצאות אחרות להתקיים.

קריאה לפעולה היא הפעלה שלה תוך כדי ריצת התוכנית. קריאה של שיגרה לעצמה ישירות או בעקיפין, נקראת רקורסיה עליה אסביר בהמשך. שגרה יכולה לקבל קלט ויכולה גם לא לקבל.
בנוסף לכך, מקובל כי שימוש מושכל בפעולות ובפונקציות עשוי לשפר את מבנה התוכנית, את קריאות הקוד ואת מידת הגמישות של התוכנית לביצוע שינויים. שימוש בפעולות מאפשר חלוקה של קוד לחלקים קצרים – שגרות קצרות - וכך מקל על וידוא נכונות של כל אחד ממרכיבי הקוד בנפרד.
תוצאה זו מאפשרת להפחית במידה משמעותית את עלויות הפיתוח והתחזוקה של תוכנה.
בתכנות מונחה-עצמים, לכל עצם יש מספר שגרות השייכות אליו, והן מגדירות את ההתנהגות של האובייקט עליו היא נקראה, ופועלות על המידע הכמוס בתוכו או בעזרתו.

בהתאם למידת החשיפה שלהן לשאר הקוד (התכונות יכולות להיות נסתרות כפי שהסברתי קודם בעקרונות השונים של התכנות מונחה העצמים). שגרות אלה מהוות למעשה ממשק בין העצם לתוכנית כולה.

ממשק -

בתכנות המושג ממשק משמש למעשה להפשטה של מחלקות התוכנה, ומגדיר את הפונקציונליות שעל מחלקות היורשות ממנו לממש כדי להיות שייכות אליו.
לדוגמא אם אובייקט עושה שימוש בממשק כלשהו, אז הדבר מבטיח שלאובייקט תהיה התנהגות מסוימת.
המימוש של ממשק הוא למעשה התחייבות של אובייקט למלא אחר סדר של דרישות והתנהגויות מסוימות.
לפיכך, ניתן להתייחס לממשק כאל חוזה, כאשר מחלקה כלשהי מממשת את כל הפונקציות המוגדרות בממשק ניתן ליצור מופע שלה, אחרת היא נחשבת מימוש אבסטרקטי – כזה שדורש הרחבה על ידי מחלקה אחרת שמשלימה את המימוש.

השימוש בממשקים הוא נהוג על מנת לשמור על כתיבה נכונה משום שבשיטה זו קיימת הפרדה בין המימוש בפועל לבין הדרישות שמאופיינות בממשק.

שימוש בProperties-

Property הינו מנגנון המאפשר לממש את עקרון הכימוס .הרעיון הוא להסתיר את המשתנים ולחשוף אותם דרך מנגנון סינון ובקרה לצורכי בדיקת תקינות המידע. בכדי להסתיר את המשתנה יש להגדיר אותו כ- private ואת ה- Property שמאפשר את הגישה למשתנה יש להגדיר כ- public.
ניתן לגשת למשתנה private אך ורק מתוך המחלקה.

שימוש בDelegate-

Delegate הוא מעין מצביע לפונקציה.
Delegate יכול לשמש כמצביע לפונקציה סטטית או פונקציה של אובייקט. באופן כזה מחלקות שונות יכולות להעביר ביניהן מצביעים אל הפונקציות שלהן.

שימוש בEvent-

תכנות מונחה האירועים הוא תפיסה בתכנות. הרעיון הוא שבתוך תוכנית המחשב קיימים חלקים

הממתינים לקבלת אות, אות הנקרא 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.
לדוגמא כשארצה ליצור מספר עצמים מסוג מסוים וארצה לעדכן את כולם בבת אחת, אוכל ליצור Event של דלגייטים (מצביעים לפונקציה) מסוג פעולת העדכון של האובייקט, ובפעולה הבונה של אותה המחלקה ממנה יצרתי את האובייקטים, להוסיף לתוך האיוונט את פעולת העדכון.
ואז, כאשר ארצה לעדכן את האובייקטים לא אצטרך לקרוא אחד אחד לפעולת העדכון אלא רק לקרוא לEvent שיתרחש.

מאפיין סטטי -

כאשר נגדיר משתנה בתוך מחלקה, הוא למעשה נוצר מספר פעמים, פעם אחת בכל אובייקט שניצור.
לעיתים נרצה להגדיר משתנים אשר מוגדרים פעם אחת לכל האובייקטים מסוג מסוים.
המשתנה הסטטי הוא משתנה שיש לו מופע אחד בלבד והוא משותף לכל המחלקה בניגוד למשתנים רגילים שמשוכפלים עבור כל אובייקט מסוג המחלקה. בנוסף לכך המשתנה הסטטי נמצא אך ורק פעם אחת בזיכרון, לא משנה כמה מופעי אובייקטים יש לאותה המחלקה.
על מנת להגדיר משתנה כסטטי יש להוסיף את המילה השמורה static בעת ההצהרה עליו.
באותה השיטה ניתן גם להגדיר כל אלמנט כסטטי כמו פרופרטי, פונקציה, מחלקה וכו'
הפונקציה הסטטית היא פונקציה שמופעלת על המחלקה כולה ולא על מופע בודד של המחלקה.

פונקציה סטטית שייכת למחלקה ולא לאובייקט, לכן בשביל לגשת לפונקציה סטטית דרך התוכנית הראשית נעשה זאת דרך שם המחלקה וציון הפונקציה בניגוד לפונקציות לא סטטיות אליהן ניגש באמצעות האובייקט של המחלקה עליו נרצה להפעיל את הפעולה.
הפעולה הסטטית לא תוכל לפנות לתכונות שאינן סטטיות מכיוון שהן יכולות להשתנות מאובייקט לאובייקט, לכן הפעולה הסטטית תוכל להשתמש אך ורק במשתנים הסטטים של המחלקה.
המחלקה הסטטית היא מחלקה ממנה לא ניתן ליצור אובייקטים, לא ניתן לרשת ממנה וכל מה שנמצא באותה המחלקה חייב להיות גם הוא סטטי.
נהוג להשתמש במחלקה הסטטית בעיקר ליצירת מחלקות עזר שיכילו פונקציות שירות.
לדוגמא Math או Console שמאפשרות פונקציות וקבועים לעזרה בחישובים מתמטיים או לעבודה עם מסך ה-Console.

אז על מה המשחק?

המשחק הינו משחק דו-ממדי עם בינה מלאכותית (Bot) שבו מתחרות ביניהן שתי דמויות.

המשחק מתחיל לאחר כחמש שניות מבניית מפת המשחק ונמשך כשתי דקות (בלי הארכות).

מטרת המשחק העיקרית היא לפגוע בכמה שיותר כדורים השייכים לך על מנת לצבור ניקוד גבוהה יותר וכמובן לא להיפסל. לכל שחקן יש מספר חיים ומספר נקודות. משחק נגמר ברגע שלאחד השחקנים נגמרים החיים או ברגע שנגמר הזמן, והמנצח נקבע לפי כמות הנקודות שצברו השניים.

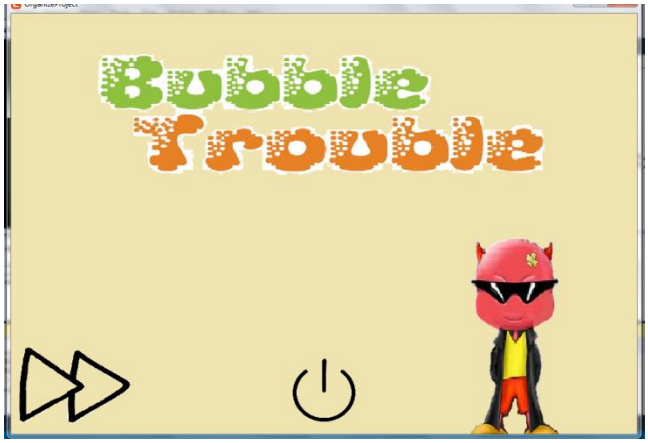
במפת המשחק ישנם כדורים בארבעה צבעים, (אדום, כחול, שחור וצהוב). הכדור האדום שייך לי השחקן ששולט בדמות בעוד שהכדור הכחול היינו של ה- Bot בהתאם. הכדור הצהוב לעומת זאת, שייך לשני השחקנים ומטרתו העיקרית לשנות את מבנה הניקוד של השחקנים ולהוסיף קצת אקשן ולהשאיר סיכוי למפסיד. כדור שחור היינו כדור מתגלגל שמופיע כל חצי דקה אשר לא ניתן לפגוע בו אלה מטרתו היא לפגוע לשחקנים ברגליים ולהוריד להם חיים.

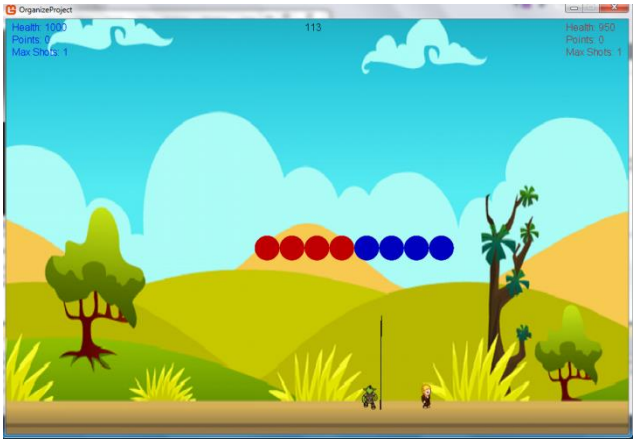
שחקן יכול לירות חץ רק בעודו עומד על הקרקע ולא זז. פגיעה בכל אחד מהכדורים (אדום, כחול וצהוב), תגרום להתפוצצות הכדור לעוד שני כדורים קטנים פי שתיים אשר ינועו בכיוונים שונים במהירות קבועה. כמובן שפגיעה של שחקן בכדור של היריב לא תפוצץ לו את הכדור.

ניתן לירות יריה רק ברגע שלא קיימת יריה באוויר. ניתן לירות כמה יריות במקביל רק כאשר תפסת קופסא שמכילה אפשרות לירות יריה אחת יותר במקביל (עד שלוש יריות במקביל).

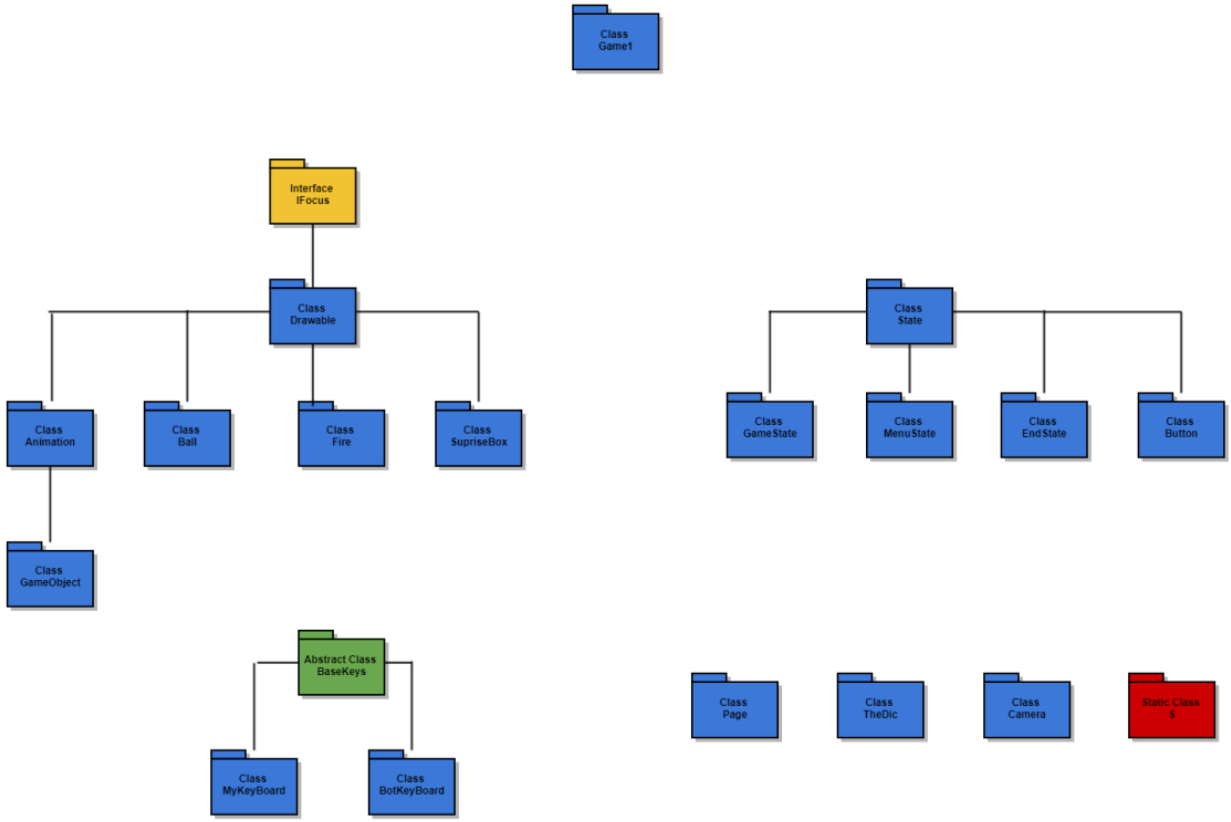
במהלך המשחק נופלות קופסאות מהשמיים. קיימים חמישה סוגי קופסאות, פלוס חיים, מינוס חיים, פלוס ניקוד, מינוס ניקוד, פלוס יריה, מקפיא את השחקן. קופסה נופלת עד לקרקע ונעלמת רק כאשר קופסה דומה הוגרלה ליפול שוב.

לשחקן 1000 חיים. פגיעה של כדור רגיל תוריד 50 חיים ותביא 50 חיים. כדור צהוב היינו כדור שמספר החיים שמוריד ומספר הניקוד שמביא ברגע הפגיעה היינו רנדומלי בין 0-100. כדור שחור יוריד 20 חיים ולא יכול להוסיף ניקוד. קופסה של ניקוד וחיים מורידה או מעלה 50 חיים/ניקוד.





תרשים UML


Class
Game1
Interface
IFocus
Class
Drawable
Class
Animation
Class
Ball
Class
Fire
Class
SupriseBox
Class
GameObject
Class
State
Class
GameState
Class
MenuState
Class
EndState
Class
Button
Abstract Class
BaseKeys
Class
MyKeyBoard
Class
BotKeyBoard
Class
Page
Class
TheDic
Class
Camera
Static Class
S

מחלקות

המחלקה הראשית – Game1

בכלליות מחלקת Game1 אחראית על מעבר המסכים ממסך התפריט למשך המשחק ולמסך סיום המשחק.

<u>מבנה המחלקה:</u>

משתנים –

- האובייקט graphics מסוג GraphicsDeviceManager שאחראי על התיאום על הכרטיס הגרפי.
- האובייקט spriteBatch מסוג SpriteBatch שבעזרתו מציירים טקסטורות על חלון המשחק.
- האובייקט Content מסוג ContentManager שבעזרתו טוענים את כל הקבצים של המשחק (טקסטורות, קבצי XML וכו'..).
- שני משתנים מסוג מחלקת State שבעזרתם נעבור בין מסכי המשחק השונים.

פעולות –

- הקונסטרקטור Game1 - שתפקידו לאתחל את המשתנים הכללים אשר בהם נשתמש פעמים רבות.

- Initialize – יכלול את ההגדרות אשר יוגדרו עם הפעלת המערכת. בין היתר הגדרות גודל המסך, הגדרת מסך מלא וכו'.

- LoadContent– בפעולה זאת כותבים את כל הקוד שתפקידו לטעון את כל הקבצים. כאן נאתחל ונשווה את המשתנים שהוגדרו סטטית במחלקה הסטטית.
בנוסף נאתחל את מילון הדמויות והגיבורים שלנו.
נעדכן את currentState לתפריט הראשי לפני התחלת המשחק.

- Update– שמקבל אובייקט מסוג GameTime קורה 60 פעמים בשנייה ובה מבצעים את הקוד שתפקידו לעדכן ולבצע פעולות בתוך המשחק.
כל עוד לא התעדכן מסך State אחר נחכה.
בפעולה זו נגדיר הגדרות כלליות ליציאה ממסך המשחק בחזרה לקוד (לחיצה על ESC).

- Draw – בפעולה זאת מציירים את הטקסטורות השונות. פעולה זאת אחראית על כל מה שרואים, כלומר כאן מצוירים המפה, העצמים והתפריטים של המשחק.

קוד -

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Collections.Generic;
using OrganizeProject.ProjectStates;

namespace OrganizeProject
{
    /// <summary>
    /// This is the main type for your game.
    /// </summary>

    public class Game1 : Game
    {
        public State currentState;
        private State nextState;

        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        public void ChangeState(State state)
        {
            nextState = state;
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before
starting to run.
        /// This is where it can query for any required services and load any
non-graphic
        /// related content.  Calling base.Initialize will enumerate through
any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            graphics.IsFullScreen = true;
            graphics.PreferredBackBufferWidth = 1000;
            graphics.PreferredBackBufferHeight = 670;
            graphics.ApplyChanges();

            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);
```

```
            S.sb = spriteBatch;
            S.gp = graphics;
            S.gd = S.gp.GraphicsDevice;
            S.cm = Content;
            S.screenWidth = GraphicsDevice.Viewport.Width;
            S.screenHeight = GraphicsDevice.Viewport.Height;
            S.font = Content.Load<SpriteFont>("SpriteFont1");
            TheDic.Init();

            currentState = new MenuState(this);
        }

        /// <summary>
        /// UnloadContent will be called once per game and is the place to
unload
        /// game-specific content.
        /// </summary>
        protected override void UnloadContent()
        {
            // TODO: Unload any non ContentManager content here
        }

        /// <summary>
        /// Allows the game to run logic such as updating the world,
        /// checking for collisions, gathering input, and playing audio.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing
values.</param>
        protected override void Update(GameTime gameTime)
        {
            if (nextState != null)
            {
                currentState = nextState;
                nextState = null;
            }

            if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
ButtonState.Pressed || Keyboard.GetState().IsKeyDown(Keys.Escape))
                Exit();

            currentState.Update(gameTime);
            currentState.PostUpdate(gameTime);

            // TODO: Add your update logic here
            base.Update(gameTime);
        }

        /// <summary>
        /// This is called when the game should draw itself.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing
values.</param>
        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.White);

            currentState.Draw(gameTime, S.sb);

            spriteBatch.End();
            base.Draw(gameTime);
        }
    }
```

מחלקת S סטטית

מחלקה זו הינה מחלקה כללית שכוללת בתוכה את הנתונים והאיפוסים הכללים שנשתמש בהם באופן קבוע מספר רב של פעמים במהלך הפרויקט. מחלקה זו היא מחלקה סטטית מכיוון שנרצה לגשת לכל המשתנים שהגדנו בה מכל מקום.

מחלקה זו היא מחלקה סטטית שלמעשה מחזיקה בכל המשתנים שכל המחלקות בפרויקט צריכות להכיר בהם וכך למעשה מאפשרת לכל מחלקה בפרויקט לגשת אליהם באמצעות S. המשתנים אותם מאכסנת המחלקה הינם משתנים קבועים, delegates, enums, ופעולות סטטיות שונות שמצריכות גישה רחבה אליהן. בתוך מחלקה זו יצרתי את הפעולה ()CreateGroundLine שלמעשה אחראית על קביעת הקרקע עליה ירוצו הדמויות ועלייה יתגלגלו הכדורים.

מבנה המחלקה:

משתנים-

- האובייקט sb מסוג SpriteBatch שאחראי על פעולת הציור הממשית של האובייקטים.
- האובייקט cm מסוג ContentManager שבעזרתו טוענים קבצים לתוך האובייקטים.
- האובייקט gp מסוג Graphics DeviceManager שאחראי על הכרטיס הגרפי.
- האובייקט gd מסוג GraphicsDevice שמכיל נתונים של הכרטיס הגרפי.
- ועוד משתנים כללים נוספים כמו זמן משחק, אובייקטים במהלך המשחק, כדורים גודל שחקן, אורך ורוחב מסך המשחק, קופסאות הפתעה נופלות ועוד..
- קיימים enum-ים של הגיבורים, פעולות הגיבורים ומהירות הפעולות.

בנוסף, במחלקה זו נגדיר את ממשק IFocus.
זהו ממשק שבהמשך נשתמש בו, הוא יהיה שימושי כאשר נרצה ליצור מצלמה שעוקבת אחרי השחקן.

```
public interface IFocus
    {
        Vector2 Position { get;}
    }
```

המטרה בשימוש בממשק היא לנוחיות, כך שכשאצור מצלמה שתקבל אובייקט היא לא תצטרך לקבל את כולו אלא רק את Focus שהוא החלק שמחזיק בפעולה שמקבלת אתPosition.המשתנה Position הוא וקטור שלמעשה מעיד על מיקום של אובייקט מסוים על המסך.

פעולות –

- פעולת ()CreateGroundLine יוצרת קרקע עלייה יוכלו השחקנים לרוץ ועליה יקפצו הכדורים במשחק. פעולה זו, בעצם, יוצרת מערך חד ממדי בגודל אורך הרצפה (Width) המוגדרת וצויירה בצייר, כל שלכל מיקום במערך(ערך x), יש ערך Y מתאים.
  כל בעצם אנו יוצריך את האדמה כאשר כל שמחק בעצם מתרחק מעלייה.
  ניעזר ב מערך צבעים כדי לקבוע את הקרקע בדרך של בדיקה שלא הגענו מחוץ לקרקע.
  Color.White =!. במהלך בניית כל הקרקע נכפיל במשתנה scale דברים מסוימים מכיוון שהגדלנו אותם בה.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Collections.Generic;
using OrganizeProject.ProjectStates;

namespace OrganizeProject
{
    public interface IFocus
    {
        Vector2 Position { get;}
    }
    public delegate void HandleCollision(GameObject obj);
    public delegate void HandleCollisionball(Ball obj);
    public delegate void HandleUpdate();
    public delegate void HandleCollisionBox(GameObject obj);
    public delegate void HandlePlayerCollision(GameObject obj);


    public enum Heros { Goblin, Man }
    public enum States { Walk, Fire, Stance }
    public enum Tempo { Slow = 7, Medium = 5, High = 2 }

    static class S
    {
        #region Data
        public static SpriteBatch sb;
        public static GraphicsDeviceManager gp;
        public static ContentManager cm;
        public static int[] ground;

        public static int screenWidth;
        public static int screenHeight;

        public static GraphicsDevice gd;
        public static Random rnd = new Random();

        public static List<Ball> balls, balls1, pointBalls;

        public static GameObject g, k;

        public static SupriseBox[] supBox;

        public static SpriteFont font;

        //public static CountdownTimer prepTimer;

        public static int i;

        public static int time = 0;

        public static float scale;

        public static int gameCDTimer = S.time + 124000;

        public static int winner = 0;

        public static GameState gameState;
```

```
        public static Ball blackBall;

        #endregion

        public static void Create_Ground_Line(float scale)
        {
            S.scale = scale;
            Texture2D floor = cm.Load<Texture2D>("Goblin/BackGround_Floor");
            ground = new int[(int)(floor.Width*scale)];
            Color[] color = new Color[floor.Width * floor.Height];
            floor.GetData<Color>(color);
            for (int i = 0; i < floor.Height; i++)
            {
                for (int j = 0; j < floor.Width; j++)
                {
                    if (color[floor.Width * i + j] != Color.White &&
color[floor.Width * (i - 1) + j] == Color.White)
                    {
                        for (int k = 0; k < scale; k++)
                        {
                            ground[(int)(j * scale) + k] = (int)(i * scale);
                        }
                    }
                }
            }
        }
    }
}
```

מחלקת Drawable

מחלקה זו היא המחלקה הבסיסית ביותר ליצירת ציור על המסך. יש בה פעולה האחראית על ציור תמונה על המסך כל עוד הציור פעיל.

מחלקת Drawable היא מחלקת בסיס לכל טקסטורה שניתן לצייר על המסך, כגון אובייקטים, כפתורים, רקע וכו'.

המחלקה הזו יורשת מהממשק Focus וזאת על מנת להבטיח שכל אובייקט מסוג MainDrawable מקיים את Focus כלומר שיש לו את הוקטור Position בשביל שאוכל להשתמש בזה אחר כך בכדי לבנות מצלמה שתוכל לעקוב אחרי אובייקטים.

במחלקת Drawable נשמר למעשה כל המידע הנחוץ על מנת לצייר טקסטורה או חלק ממנה על המסך. מידע לדוגמה : Texture, Rectangle, Scale וכו'. המחלקה מחזיקה במספר פעולות בונות ובכך מתאימה את עצמה לכמות המידע שיש לצייר על המסך.

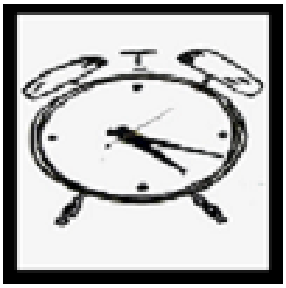
במחלקה זו קיימות המון תכונות. כל אובייקט שמצויר בעצם ויורש את מחלקת זו, בעל תכונות אלה. לא כל אובייקט שמצויר משתמש בכל התכונות אך לכולם יש אותן.

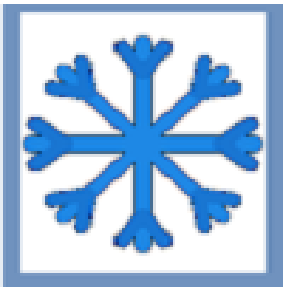
נעדכן לתכונות Get Set כדי לאפשר גישה ועדכון לתכונות אלו במחלקות אחרות.

```
public Texture2D Texture { get; set; }
```

פעולות –

- פעולת Draw() אשר מאפשרת ציור באמצעות SpriteBatch לה נשלח את כל התכונות.
פעולה זו מוגדרת כ virtual על מנת לאפשר את גריסתה (override)בכל מחלקה שיורשת.





```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;

namespace OrganizeProject
{
    public class Drawable : IFocus
    {
        #region Data
        public Texture2D Texture { get; set; }
        public Rectangle? SourceRectangle { get; set; }
        public Rectangle destinationRectangle { get; set; }
        public Vector2 Origin { get; set; }
        public SpriteEffects Effects { get; set; }
        public Vector2 Position { get; set; }
        public Color color { get; set; }
        public float rotation { get; set; }
        public Vector2 scale { get; set; }
        public float layerDepth { get; set; }

        public bool isDraw { get; set; }

        public SpriteFont font { get; set; }
        public string text { get; set; }

        #endregion

        #region Ctors
        public Drawable(Vector2 position, Color color, float rotation, Vector2
scale, SpriteEffects effects, float layerDepth)
        {
            this.Position = position;
            this.color = color;
            this.rotation = rotation;
            this.scale = scale;
            this.Effects = effects;
            this.layerDepth = layerDepth;
            isDraw = true;
        }

        public Drawable(Texture2D texture, Vector2 position, Rectangle?
sourceRectangle,
                        Color color, float rotation, Vector2 origin, Vector2
scale, SpriteEffects effects, float layerDepth)
        {
            Texture = texture;
            this.Position = position;
            SourceRectangle = sourceRectangle;
            this.color = color;
            this.rotation = rotation;
            Origin = origin;
            this.scale = scale;
            this.Effects = effects;
            isDraw = true;
            this.layerDepth = layerDepth;

        }
```

```
        public Drawable(Color color, float rotation, Vector2 scale, float
layerDepth)
        {
            this.color = color;
            this.rotation = rotation;
            this.scale = scale;
            this.layerDepth = layerDepth;
            isDraw = true;

        }

        public Drawable(Vector2 position, Color color, Vector2 scale, float
layerDepth, Texture2D texture, Rectangle destinationRectangle)
        {
            this.Position = position;
            this.color = color;
            this.scale = scale;
            this.layerDepth = layerDepth;
            this.Texture = texture;
            this.destinationRectangle = destinationRectangle;
            isDraw = true;
        }

        public Drawable(Vector2 position, Color color)
        {
            this.Position = position;
            this.color = color;

            isDraw = true;
        }

        public Drawable(Vector2 position, Color color, SpriteFont font)
        {
            Position = position;
            this.color = color;
            this.font = font;
            this.text = text;
        }
        #endregion

        public virtual void Draw()
        {
            if (isDraw && Texture != null)
            {
                S.sb.Draw(
                        Texture, Position, SourceRectangle,
                        color, rotation, Origin,
                        scale, Effects, layerDepth);
            }
        }
    }
}
```

מחלקת Page

במחלקה זו נמצא מידע עבור מצב מסוים של גיבור מסוים.

מידע זה מכיל את המידע הנחוץ על מנת לצייר את האנימציה של אותו גיבור במצב המסוים וגם מידע על המקומות בהם אפשר להתנגש עם הדמות במצב הנתון.

רקע לפני גישה לפעולת ה Page -

על מנת להפריד את הפעולות והתזוזות השונות של דמות בחרנו תמונת spritesheet, כלומר תמונת אנימציה ארוכה של דמות באותו מצב עם שינוי קטן. לכל פעולה מספר פריימים.

על כל תמונת אנימציה ציירנו בשורה האחרונה בתחתית התמונה נקודות בצבע השונה מצבע הרקע מאחור. נקודה בכל צד של מצב הדמות כדי לתחום אותה ב Rectangle, ועוד נקודה במרכז ה Rectangle שבעצם מבטא origin, נקודת היסט.

את ה פריימים ונקודות ההיסט נכניס לList.

התוכנית משתמשת במחלקה אשר מטפלת באנימציה ושמה Animation עליה אסביר בהמשך ובאמצעות פעולת העדכון שלה התוכנית מריצה את המצבים השונים של הדמות בהתאם לבחירת המשתמש.

פעולות-

- פעולת FindTempo(). עבור כל אנימציה יש מהירות מסוימת שבה עוברים התמונות, דרך פונקציית ChooseTempo אנו משיגים את מהירות זו.
- פעולת MakeTransparent() כדי להפוך את הרקע לשקוף.

קוד -

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Linq;
using System.Collections.Generic;

namespace OrganizeProject
{
    public class Page
    {
        #region Data
        public Texture2D Tex { get; private set; }
        public List<Rectangle> Recs { get; private set; }
        public List<Vector2> Orgs { get; private set; }
        public Tempo Pace { get; private set; }
        #endregion
```

```
        #region Ctors
        public Page(Heros hero, States state)
        {
            Recs = new List<Rectangle>();
            Orgs = new List<Vector2>();
            Tex = S.cm.Load<Texture2D>(hero + "/" + state);
            Color[] c = new Color[Tex.Width];
            List<int> pos = new List<int>();

            Tex.GetData(0, 0, new Rectangle(0, Tex.Height - 1, Tex.Width, 1),
c, 0, Tex.Width);
            for (int i = 0; i < c.Length; i++)
            {
                if (c[i] != c[1])
                    pos.Add(i);
            }
            for (int i = 0; i < pos.Count - 2; i += 2)
            {
                Recs.Add(new Rectangle(pos[i], 0, pos[i + 2] - pos[i],
Tex.Height - 2));
            }
            for (int i = 0; i < pos.Count - 2; i += 2)
            {
                Orgs.Add(new Vector2(pos[i + 1] - pos[i], Tex.Height - 2));
            }
            Pace = FindTempo(state);

            // Make background color transparent
            MakeTransparent();
        }
        #endregion

        private Tempo FindTempo(States state)
        {
            switch (state)
            {
                case States.Stance:
                    return Tempo.Slow;
                case States.Walk:
                    return Tempo.Slow;
                case States.Fire:
                    return Tempo.Slow;
                default:
                    return Tempo.Slow;
            }
        }

        private void MakeTransparent()
        {
            Color[] allcolor = new Color[Tex.Width * Tex.Height];

            Tex.GetData<Color>(allcolor);
            for (int i = 1; i < allcolor.Length; i++)
            {
                if (allcolor[i] == allcolor[0])
                    allcolor[i] = Color.Transparent;
            }
            allcolor[0] = Color.Transparent;
            Tex.SetData<Color>(allcolor);
        }
    }
```

מחלקת TheDic

מחלקה זו היא בעצם מילון. מילון מורכב מדפים Page. כל Page היינו מצב של דמות מסוימת.

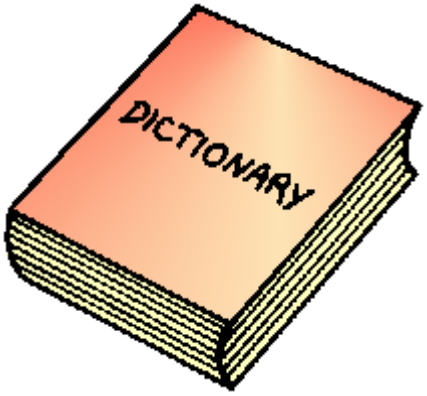
מטרת המילון לרכז את כל אותם דפים באופן של גיבור ופעולותיו.

בעצם המילון היינו מילון לפי גיבורים שלכל גיבור יש מילון של פעולות לפי דפים.

כל זה נוצר בפעולת Init() של המחלקה.

קוד -

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Linq;
using System.Collections.Generic;
using System.IO;

namespace OrganizeProject
{
    static class TheDic
    {
        public static Dictionary<Heros, Dictionary<States, Page>> Big =
            new Dictionary<Heros, Dictionary<States, Page>>();

        public static void Init()
        {
            foreach (Heros hero in Enum.GetValues(typeof(Heros)))
            {
                Dictionary<States, Page> heroDic = new Dictionary<States,
Page>();
                foreach(States state in Enum.GetValues(typeof(States)))
                {
                    if (File.Exists(Directory.GetCurrentDirectory() +
"/Content/" + hero + "/" + state + ".xnb"))
                    {
                        heroDic.Add(state, new Page(hero, state));
                    }
                }
                Big.Add(hero, heroDic);
            }

        }

    }
}
```



מחלקת Animation

מחלקה זו יורשת מ – Drawable. מחלקה זו אחראית על יצירת אנימציה על המסך. היא כל כמה זמן משנה פריים ומציירת אותו וכך רואים הנפשה על המסך. לכל דמות ישנה אנימציה שונה.

פעולות -

- בנאים המאתחלים את האנימציה לפריים הראשון ומעדכנים למצב הנבחר של אותה הדמות.
- פעולת Update() - עדכון לפריים הבא או חזרה לראשון בהתאם לתכונות ולפעולות על המחלקה. ברגע שהשחקן מחליף כיוון צריך לשנות לו את נקודת הOrigin.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Linq;
using System.Collections.Generic;

namespace OrganizeProject
{
    public class Animation : Drawable
    {
        #region Data
        public Page Anime { get; set; }
        public int CurrentIndex { get; set; }
        public int CurrentFrame { get; set; }
        #endregion

        #region Ctors
        public Animation(Heros hero, States state, Vector2 position, Color
color, float rotation, Vector2 scale, SpriteEffects effects, float layerDepth)
:
                        base(position, color, rotation, scale, effects,
layerDepth)
        {
            Anime = TheDic.Big[hero][state];
            CurrentIndex = 0;
            CurrentFrame = 0;
            Texture = Anime.Tex;
        }
        public Animation(Heros hero, States state, Vector2 position, Color
color, float rotation, Vector2 scale, float layerDepth) :
                        base(color, rotation, scale, layerDepth)
        {
            Anime = TheDic.Big[hero][state];
            CurrentIndex = 0;
            CurrentFrame = 0;
            Texture = Anime.Tex;
        }

        public Animation(Heros hero, States state, Vector2 position, float
rotation, Vector2 scale, float layerDepth) :
                        base(Color.White, rotation, scale, layerDepth)
        {
            Anime = TheDic.Big[hero][state];
            CurrentIndex = 0;
            CurrentFrame = 0;
            Texture = Anime.Tex;
        }
        #endregion

        public void Update()
        {
            Texture = Anime.Tex;
            SourceRectangle = Anime.Recs[CurrentIndex];
            if (Effects == SpriteEffects.FlipHorizontally)
                Origin = new Vector2(SourceRectangle.Value.Width -
Anime.Orgs[CurrentIndex].X, Anime.Orgs[CurrentIndex].Y);
            else
                Origin = Anime.Orgs[CurrentIndex];
            if ((int)Anime.Pace < CurrentFrame++)
```

```
            {
                CurrentFrame = 0;
                CurrentIndex++;
                CurrentIndex %= Anime.Recs.Count;
            }

            base.Draw();
        }
    }
}
```

מחלקת GameObject

מחלקה זו אחראית על הדמויות במשחק. דמות אחת היא החשקן אותו אנו משחקים ואילו הדמות השנייה היינה בינה מלאכותית. לכל שחקן מספר חיים ומספר יריות קבועים שאיתם הוא מתחיל את המשחק. במהלך המשחק השחקן יכול לצבור קופסאות הפתעה שיכולות לשנות את מאזן הנקודות החיים וכו'.

כל שחקן יכול ללכת או לרוץ(ימינה או שמאלה) לקפוץ ולרוץ בהתאם למחלקת ה Animation ו- MyKeyBoard.

פעולות –

- במחלקה זו קיים בנאי אחד. אשר בו נעדכן את כל הקשור לשחקן כגון חיים, זמן הקפאה, יריות, מהירות, גרביטציה ועוד. בבנאי נבנה את היריה של השחקן. יריה שייכת לשחקן אחד ובגודל פיקסל על פיקסל.
- פעולת UpdateState() אחראית על עדכון השחקן. נגדיר שכדורים יוכלו לפגוע בשחקן רק לאחר שנייה מרגע הפגיעה האחרונה. נגדיר זמן הקפאה ברגע תפיסת הקופסה המתאימה. בנוסף, נעדכן את פעולת העידכון של מחלקת האנימציה.
- פעולת UpdateState() מבצעת בתוכה גם את פעולת Input(). פעולה זו אחראית לכל תזוזות השחקן. בפעולה זו נגדיר שלא ניתן לצאת מגבולות המסך.
- פעולת UpdateState() מבצעת גם את פעולת Move(). פעולה זו מפעילה את פעולת Gravity() שאסביר בהמשך. בנוסף, פעולה זו מעדכנת את מיקום השחקן לפי מהירות שהשתנתה במהלך פעולת UpdateState() ונאפס בחזרה את המהירות.
- פעולת Gravity() מעדכנת את נפילת השחקן ברגע שאינו נמצא על הרצפה.
- פעולת AddFire() מוסיפה עוד יריה לשחקן. מקסימום יריות במקביל היינו 3.

קוד –

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Linq;
using System.Collections.Generic;
using OrganizeProject.ProjectStates;

namespace OrganizeProject
{
    public class GameObject : Animation
    {
        #region Data
        public Heros hero;
        public BaseKeys keyboard;
        Vector2 drc;
        float gravitaion;
```

```
        Fire fire;
        Texture2D pointTex;
        public Color ballColor;
        public int count1, countPts;
        public bool isHit;
        public int lastHitTime;
        public int hitDelay = 1000;

        public int fireCount;
        public int maxFireCount;
        public int freezeTime;
        int shootDelay = 500;
        int lastShotTime;

        float velocityX;
        float velocityY;

        float accelerationY;

        float jumpVelocity;
        List<Fire> playerShots;

        //public int redTime;

        #endregion

        #region Ctors
        public GameObject(Heros hero, Vector2 position, Color color, float
rotation, Vector2 scale, float layerDepth,BaseKeys keyboard )
            : base(hero, States.Stance, position, rotation, scale, layerDepth)
        {
            this.hero = hero;
            this.keyboard = keyboard;        Position = position;

            gravitaion = 0.5f;

            ballColor = color;

            pointTex = new Texture2D(S.gd, 1, 1);
            pointTex.SetData(new Color[] { Color.White });
            fire = new Fire(ballColor, pointTex, position, null, Color.Black,
MathHelper.Lerp(0, 0.01f * (float)S.rnd.NextDouble(), 1f), new Vector2(0.5f,
1), new Vector2(1f), SpriteEffects.None, 1, this);
            fire.isDraw = false;
            playerShots = new List<OrganizeProject.Fire>();
            playerShots.Add(fire);
            count1 = 1000;
            countPts = 0;
            fireCount = 0;
            maxFireCount = 1;
            freezeTime = 0;
            lastShotTime = 0;

            drc = new Vector2(1, 1);
            accelerationY = 0.16f;
            velocityX = 0;
            velocityY = 0;
            jumpVelocity = -5f;

            GameState.PlayerCollision_Event += PlayerCollision;

        }
```

```
        #endregion

        public void UpdateState()
        {
            if (S.time - lastHitTime > 1000)
                this.color = Color.White;


            if (freezeTime == 0)
            {
                Input();
                Move();

            }
            else
            {
                if (freezeTime == S.time)
                    freezeTime = 0;
            }

            base.Update();

            for (int i = 0; i < playerShots.Count; i++)
            {
                if (playerShots[i].isDraw)
                    playerShots[i].Draw();
            }
        }

        private void Input()
        {
            if (keyboard.LeftPressed())
            {
                if (keyboard.ShiftPressed())
                {
                    if ((Anime != TheDic.Big[hero][States.Walk] || Effects ==
SpriteEffects.None))
                    {
                        CurrentIndex = 0;
                        CurrentFrame = 0;
                        Effects = SpriteEffects.FlipHorizontally;
                    }
                    velocityX = -6;
                    Anime = TheDic.Big[hero][States.Walk];


                    if (keyboard.UpPressed() && Position.Y ==
S.ground[(int)Position.X])
                    {
                        velocityY = jumpVelocity;
                    }
                }
                else
                {
                    if ((Anime != TheDic.Big[hero][States.Walk] || Effects ==
SpriteEffects.None))
                    {
                        CurrentIndex = 0;
                        CurrentFrame = 0;
                        Effects = SpriteEffects.FlipHorizontally;
                    }
                    velocityX = -5;
```

```
                    Anime = TheDic.Big[hero][States.Walk];

                    if (keyboard.UpPressed() && Position.Y ==
S.ground[(int)Position.X])
                    {
                        velocityY = jumpVelocity;
                    }
                }
            }
            else if (keyboard.RightPressed())
            {
                if (keyboard.ShiftPressed())
                {
                    if ((Anime != TheDic.Big[hero][States.Walk] || Effects ==
SpriteEffects.FlipHorizontally))
                    {
                        CurrentIndex = 0;
                        CurrentFrame = 0;
                        Effects = SpriteEffects.None;
                    }
                    velocityX = 6;
                    Anime = TheDic.Big[hero][States.Walk];

                    if ( keyboard.UpPressed() && Position.Y ==
S.ground[(int)Position.X])
                    {
                        velocityY = jumpVelocity;
                    }
                }
                else
                {
                    if ((Anime != TheDic.Big[hero][States.Walk] || Effects ==
SpriteEffects.FlipHorizontally))
                    {
                        CurrentIndex = 0;
                        CurrentFrame = 0;
                        Effects = SpriteEffects.None;
                    }
                    velocityX = 5;
                    Anime = TheDic.Big[hero][States.Walk];

                    if (keyboard.UpPressed() && Position.Y ==
S.ground[(int)Position.X])
                    {
                        velocityY = jumpVelocity;
                    }
                }
            }
            else if (((keyboard.SpacePressed()) && Anime !=
TheDic.Big[hero][States.Fire]) && S.time - lastShotTime > shootDelay)
            {
                lastShotTime = S.time;
                CurrentIndex = 0;
                CurrentFrame = 0;
                Anime = TheDic.Big[hero][States.Fire];
                Effects = SpriteEffects.None;
                if (Position.Y == S.ground[(int)Position.X] && fireCount <
maxFireCount && playerShots[fireCount].isDraw == false)
                {
                    playerShots[fireCount].Position = Position;
                    playerShots[fireCount].isDraw = true;
                    playerShots[fireCount].ccc = 0;
```

```
                    fireCount++;
                }

                if (keyboard.UpPressed() && Position.Y ==
S.ground[(int)Position.X])
                {
                    velocityY = jumpVelocity;
                }
            }
            else if (keyboard.UpPressed() && Position.Y ==
S.ground[(int)Position.X])
            {
                velocityY = jumpVelocity;
            }
            else if (Anime != TheDic.Big[hero][States.Stance])
            {
                Anime = TheDic.Big[hero][States.Stance];
                CurrentIndex = 0;
                CurrentFrame = 0;
            }
            Vector2 pos = Position;

            if (Position.X + 5 * velocityX >= S.screenWidth * S.scale)
                pos.X = S.screenWidth * S.scale - 5* velocityX;

            if (Position.X + 5 * velocityX <= 0)
                pos.X = -5 * velocityX;

            Position = pos;

        }

        private void Move()
        {
            Gravity();

            Position += new Vector2(drc.X * velocityX, drc.Y * velocityY);
            velocityX = 0;

        }



        private void Gravity()
        {
            if (this.Position.Y + velocityY < S.ground[(int)Position.X])
            {
                velocityY += accelerationY;
            }
            else
            {
                Position = new Vector2(Position.X, S.ground[(int)Position.X]);
                velocityY = 0;
            }
        }

        public void AddFire()
        {
            Fire fire2 = new Fire(ballColor, pointTex, Position, null,
Color.Black, MathHelper.Lerp(0, 0.01f * (float)S.rnd.NextDouble(), 1f), new
Vector2(0.5f, 1), new Vector2(1f), SpriteEffects.None, 1, this);
            fire2.isDraw = false;
```

```
            playerShots.Add(fire2);
            if (maxFireCount < 3)
                maxFireCount++;
        }

        public void PlayerCollision(GameObject obj)
        {
            Rectangle myRectangle = destinationRectangle;
            destinationRectangle = myRectangle;
            if (myRectangle.Intersects(obj.destinationRectangle))
            {
                if (myRectangle.X + myRectangle.Width >= obj.Position.X)
                {
                    myRectangle.X = (int)obj.Position.X + myRectangle.Width;
                }

                if (myRectangle.X + myRectangle.Width <= obj.Position.X)
                {
                    myRectangle.X = (int)obj.Position.X - myRectangle.Width;
                }
            }

        }
    }
}
```

מחלקת Ball

המחלקה אחראית על כל הכדורים הנמצאים על המסך. לכל כדור מהירות בשני הצירים וצבע ייחודי.

פעולות –

- במחלקה זו מספר בנאים בשל היות הכדורים שונים בגודלם בצבעם ובמהירותם.
  נגדיר מהירות. נגדיר את אירוע ה Collision בכל בנאי.

- פעולת ()UpdateBall אחראית על עדכון הכדור.

- פעולת ()Update בעצם אחראית על שינוי הכיוון ברגע פגיעה בגבולות המסך.

- פעולת ()Collision בודקת פגיעת כדור בשחקן. ברגע פגיעה של הכדור בשחקן הכדור מופנה ומועף, (מהירות שלילית). במקרה כאן נבדוק מקרי קצה של פגיעה בקצוות השחקן משני הצדדים. ביצעתי בדיקה שבה אם כדור מגיע מימין ופוגע בקצה השמאלי של השחקן הוא ימשיך לנוע לאותו כיוון ולהפך. זמן פגיעה אחרון בשחקן מתעדכן, יורדים לשחקן חיים והוא נצבע באדום לכמה רגעים.

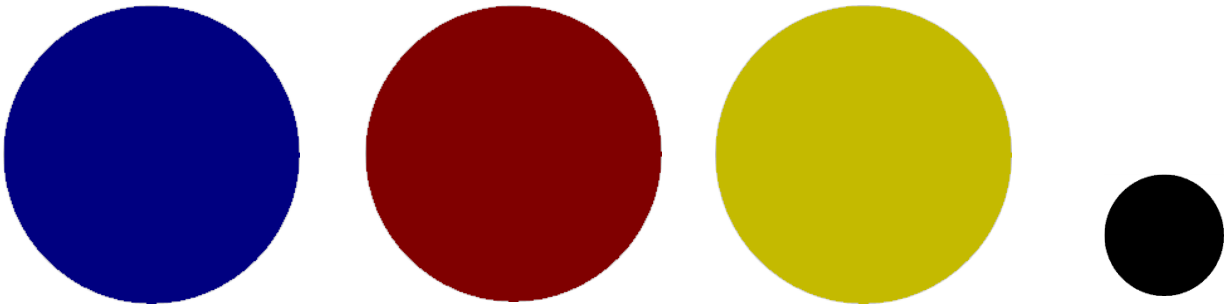
- נגרוס את פעולת ()Draw על מנת לצייר בדרך שונה מהנתון בDrawable.

קוד -

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using OrganizeProject.ProjectStates;

namespace OrganizeProject
{
    public class Ball : Drawable
    {
        #region Data
        float gravitaion;
        public Vector2 velocity;
        public int points, health;
        Color uniqueColor;
        public int countHitsWall = 0;
        #endregion

        #region Ctors
        public Ball(Vector2 position, Color color, Vector2 scale, float
layerDepth, Texture2D Texture, Rectangle Rectangle, int points, int health)
            : base(position, color, scale, layerDepth, Texture, Rectangle)
        {
            Position = position;
            gravitaion = 0.5f;
            velocity.X = -5f;
            velocity.Y = -10f;
            this.points = points;
            this.health = health;
            this.uniqueColor = color;
```

```
            countHitsWall = 0;

            GameState.Collision_Event += Collision;
        }
        public Ball(Vector2 position, Color color, Vector2 scale, float
layerDepth, Texture2D Texture, Rectangle Rectangle, Vector2 velocity, int
points, int health)
            : base(position, color, scale, layerDepth, Texture, Rectangle)
        {
            Position = position;
            gravitaion = 0.5f;
            this.velocity.X = velocity.X;
            this.velocity.Y = velocity.Y;
            this.points = points;
            this.health = health;
            this.uniqueColor = color;
            countHitsWall = 0;

            GameState.Collision_Event += Collision;
        }
        #endregion

        public void UpdateBall()
        {
            Update();
        }


        public void Update()
        {
            Rectangle myRectangle = destinationRectangle;
            myRectangle.X = myRectangle.X + (int)velocity.X;
            myRectangle.Y = myRectangle.Y + (int)velocity.Y;
            destinationRectangle = myRectangle;

            if (myRectangle.X <= 0)
            {
                velocity.X = -velocity.X;
                myRectangle.X = 0;
                countHitsWall++;
            }

            if (myRectangle.X + myRectangle.Width >= S.screenWidth * S.scale)
            {
                velocity.X = -velocity.X;
                myRectangle.X = S.screenWidth - myRectangle.Width;
                countHitsWall++;
            }

            if (myRectangle.Y <= 0)
            {
                velocity.Y = -velocity.Y;
                myRectangle.Y = 0;
            }
            if (myRectangle.Y + myRectangle.Height >= S.ground[myRectangle.X])
            {
                velocity.Y = -velocity.Y;
                myRectangle.Y = S.ground[myRectangle.X] - myRectangle.Height;
            }


            Draw();
```

```
        }
        public void Collision(GameObject obj)
        {
            if (obj.isDraw == true && isDraw)
            {
                Texture2D myTex = Texture;
                Rectangle myRectangle = destinationRectangle;
                Rectangle playerrec = new Rectangle((int)(obj.Position.X -
obj.Origin.X), (int)(obj.Position.Y - obj.Origin.Y), obj.Anime.Recs[0].Width,
obj.Anime.Recs[0].Height);
                if (myRectangle.Intersects(playerrec) && S.time -
obj.lastHitTime >= obj.hitDelay)
                {
                    if (destinationRectangle.X + destinationRectangle.Width / 2
* scale.Length() <= obj.Position.X - obj.Origin.X)
                    {
                        velocity.X = -Math.Abs(velocity.X);
                    }
                    else if (destinationRectangle.X >= obj.Position.X +
obj.Origin.X)
                    {
                        velocity.X = Math.Abs(velocity.X);
                    }
                    else
                        velocity.Y = -Math.Abs(velocity.Y);


                    obj.lastHitTime = S.time;


                    if (obj.ballColor == color || uniqueColor == Color.Yellow
|| uniqueColor == Color.Black)
                    {
                        if (obj.count1 - this.health > 0)
                            obj.count1 -= this.health;
                        else
                            obj.count1 = 0;

                        obj.color = Color.Red;
                    }
                }
            }
        }



        public override void Draw()
        {
            if(isDraw)
                S.sb.Draw(Texture, destinationRectangle, color);
            //base.Draw();
        }
    }
}
```



מחלקת Fire

מחלקה זו אחראית על הירייה היוצאת מהשחקן ברגע לחיצה על מקש הירייה.

ירייה היא בעצם פיקסל אחד אשר גדל בקצב קבוע בפעולה. נגדיר את נקודת ה Origin שלו למטה במרכז ובכך נגדיל את הירייה כלפי מעלה בלבד.

פעולות -

- נגדיר בנאי קבוע בו נאפס את הירייה. נגדיר את אירועי ה- Collision בבנאי של רגע הפגיעה של הירייה בכדור ואירוע עדכון גודל הירייה.

- פעולת בדיקת הפגיעה של ירייה בכדור CheckCollision(). אם ירייה מגיעה על לסוף המסך בציר Y היא תיעלם ותהיה אפשרות לירות עוד ירייה. כל שחקן יכול לפגוע רק בכדורים השייכים לצבע שלו ובכדור הצהוב המיוחד.

ברגע הפגיעה מתבצע פיצול לשני כדורים קטנים פי שתיים אשר ינועו במהירות לשני כיוונים נגדיים. אם גודל הכדור קטן מספיק הוא יעלם לגמרי. ברגע הפגיעה יעלה ניקוד לשחקן שפגע.

- פעולת Update() אשר אחראית על הגדלת הכדור אנכית ברגע שירייה נורתה.

קוד -

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Collections.Generic;
using OrganizeProject.ProjectStates;

namespace OrganizeProject
{
    class Fire : Drawable
    {
        Color BallColor;
        GameObject shooter;
        public float ccc;


        public Fire(Color ballColor, Texture2D texture, Vector2 position,
Rectangle? sourceRectangle,
                        Color color, float rotation, Vector2 origin, Vector2
scale, SpriteEffects effects, float layerDepth, GameObject shooter)
            : base(texture, position, sourceRectangle, color, rotation, origin,
new Vector2(5, 0), effects, layerDepth)
        {
```

```
            BallColor = ballColor;
            isDraw = false;
            ccc = 0;
            this.shooter = shooter;

            GameState.Update_event += Update;
            GameState.FireCollision_Event += CheckCollision;
        }

        public void CheckCollision(Ball ball)
        {
            Rectangle rec = new Rectangle((int)(Position.X - scale.X),
(int)(Position.Y - scale.Y), (int)(Texture.Width * scale.X),
(int)(Texture.Height * scale.Y));

            if (isDraw)
            {
                if (Position.Y - scale.Y <= 0)
                {
                    isDraw = false;
                    ccc = 0;
                    shooter.fireCount--;
                }
                if (rec.Intersects(ball.destinationRectangle) && isDraw)
                {
                    isDraw = false;
                    shooter.fireCount--;
                    ccc = 0;
                    if (ball.color == this.BallColor)
                    {
                        ball.isDraw = false;
                        if (ball.color == Color.Red)
                        {
                            S.balls.Remove(ball);
                            S.g.countPts += ball.points;
                            if ((int)(0.5 * ball.destinationRectangle.Width) >
5)
                            {
                                S.balls.Add(new Ball(ball.Position, BallColor,
new Vector2(1f), 0, ball.Texture, new
Rectangle((int)ball.destinationRectangle.X, (int)ball.destinationRectangle.Y,
(int)(0.5 * ball.destinationRectangle.Height), (int)(0.5 *
ball.destinationRectangle.Width)), 50, 50));
                                S.balls.Add(new Ball(ball.Position, BallColor,
new Vector2(1f), 0, ball.Texture, new
Rectangle((int)ball.destinationRectangle.X, (int)ball.destinationRectangle.Y,
(int)(0.5 * ball.destinationRectangle.Height), (int)(0.5 *
ball.destinationRectangle.Width)), new Vector2(5f, -10f), 50, 50));
                            }
                        }
                        else if (ball.color == Color.Blue)
                        {
                            S.balls1.Remove(ball);
                            S.k.countPts += ball.points;
                            if ((int)(0.5 * ball.destinationRectangle.Width) >
5)
                            {
                                S.balls1.Add(new Ball(ball.Position, BallColor,
new Vector2(1f), 0, ball.Texture, new
Rectangle((int)ball.destinationRectangle.X, (int)ball.destinationRectangle.Y,
(int)(0.5 * ball.destinationRectangle.Height), (int)(0.5 *
ball.destinationRectangle.Width)), 50, 50));
```

```
                                S.balls1.Add(new Ball(ball.Position, BallColor,
new Vector2(1f), 0, ball.Texture, new
Rectangle((int)ball.destinationRectangle.X, (int)ball.destinationRectangle.Y,
(int)(0.5 * ball.destinationRectangle.Height), (int)(0.5 *
ball.destinationRectangle.Width)), new Vector2(5f, -10f), 50, 50));
                            }
                        }


                    }
                    else if (ball.color == Color.Yellow)
                    {
                        ball.isDraw = false;
                        S.pointBalls.Remove(ball);
                        shooter.countPts += ball.points;

                        if ((int)(0.5 * ball.destinationRectangle.Width) > 5)
                        {
                            S.pointBalls.Add(new Ball(ball.Position,
ball.color, new Vector2(1f), 0, ball.Texture, new
Rectangle((int)ball.destinationRectangle.X, (int)ball.destinationRectangle.Y,
(int)(0.5 * ball.destinationRectangle.Height), (int)(0.5 *
ball.destinationRectangle.Width)), S.rnd.Next(0, 100), S.rnd.Next(0, 100)));
                            S.pointBalls.Add(new Ball(ball.Position,
ball.color, new Vector2(1f), 0, ball.Texture, new
Rectangle((int)ball.destinationRectangle.X, (int)ball.destinationRectangle.Y,
(int)(0.5 * ball.destinationRectangle.Height), (int)(0.5 *
ball.destinationRectangle.Width)), new Vector2(7f, -12f), S.rnd.Next(0, 100),
S.rnd.Next(0, 100)));
                        }
                    }
                }
            }
        }

        public void Update()
        {
            if (isDraw)
            {
                scale = new Vector2(3, ccc += 10f);
            }
            else
            {
                ccc = 0;
            }
        }
    }
}
```

מחלקת SupriseBox

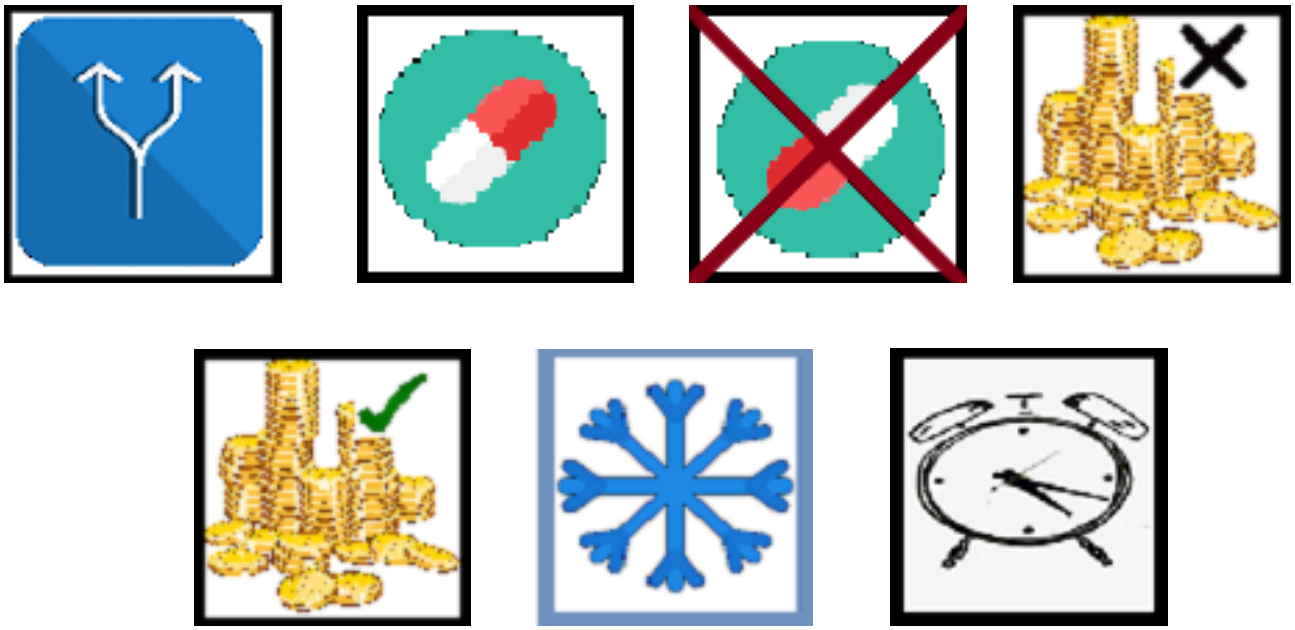
מחלקה זו אחראית על הקופסאות שנופלות מידי כמה שניות מראש המסך כלפי מעלה בגרביטציה.

כל קופסה בעלת תפקיד שונה. קיימת קופסה להוספת או הפחתת חיים, להוספת או הפחתת ניקוד, להקפאת השחקן שתופס את הקופסה ולהארכת זמן המשחק.

קופסה שנופלת על הרצפה נשארת עלייה וניתן לתפוס אותה כל עוד קופסה דומה לא רוצה ליפול שוב. המטרה למנוע משתי קופסאות דומות להיות במסך במקביל.

פעולות-

- קיים בנאי לאיפוס המשתנים.
- קיימת פעולת UpdateBox() אשר אחראית על עדכון הכדור. הקופסה נופלת בגרביטציה מסוימת כל עוד לא מגיעה לרצפה S.Ground . ברגע ההגעה הקופסה נעצרת עד שלא נתפסה על ידי אחד השחקנים או עד שלא הופיעה קופסה זהה.
- פעולת Collision() אשר שולטת על פגיעה השחקן בקופסה ועדכון התנותים בהתאם. תפיסת קופסה תצבע את השחקן לצהוב לכמה רגעים.



קוד -

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Linq;
using System.Collections.Generic;
using OrganizeProject.ProjectStates;

namespace OrganizeProject
{
    class SupriseBox : Drawable
    {
        #region Data
        float gravitaion;
        Vector2 drc;
        int timeDisappearBox;
        public int numOfBox;
        #endregion

        #region Ctors
        public SupriseBox(Vector2 position, Color color, Vector2 scale, float
layerDepth, Texture2D Texture, Rectangle Rectangle, int numOfBox)
            : base(position, color, scale, layerDepth, Texture, Rectangle)
        {
            Position = position;
            gravitaion = 0.01f;
            drc = Vector2.Zero;
            this.numOfBox = numOfBox;
            timeDisappearBox = S.gameCDTimer - 5000;

            GameState.BoxCollision_Event += Collision;
        }
        #endregion

        public void UpdateBox()
        {
            Move();
            drc = Vector2.Zero;
        }
        private void Move()
        {
            if (isDraw)
            {
                Gravity();
            }
            Position += drc;
        }

        private void Gravity()
        {
            drc.Y += gravitaion;
            gravitaion += 0.01f;
            if (Position.Y + (Texture.Height * scale.Length()) >=
S.ground[(int)Position.X])
            {
                gravitaion = 0;
                if ((timeDisappearBox - S.time) / 1000 == 5)
                    isDraw = false;
            }
```

```
        }

        public void Collision(GameObject obj)
        {
            if (obj.isDraw == true && isDraw)
            {
                Rectangle myRectangle = new Rectangle((int)Position.X,
(int)Position.Y, (int)(Texture.Width * scale.Length()), (int)(Texture.Height*
scale.Length()));
                Rectangle playerrec = new Rectangle((int)(obj.Position.X -
obj.Origin.X), (int)(obj.Position.Y - obj.Origin.Y), obj.Anime.Recs[0].Width,
obj.Anime.Recs[0].Height);

                if (myRectangle.Intersects(playerrec))
                {

                    isDraw = false;
                    gravitaion = 0;

                    if (numOfBox == 0)
                        obj.AddFire();
                    else if (numOfBox == 1)
                        obj.count1 += 50;
                    else if (numOfBox == 2)
                        obj.countPts += 50;
                    else if (numOfBox == 3)
                        S.gameCDTimer += 20000;
                    else if (numOfBox == 4)
                        obj.freezeTime = S.time + 5000;
                    else if (numOfBox == 5)
                        obj.count1 -= 50;
                    else if (numOfBox == 6)
                        obj.countPts -= 50;

                    obj.color = Color.Yellow;
                }
            }
        }
    }
}
```

מחלקת MyKeyBoard

המחלקה האבסטרקטית BaseKeys

מחלקה אבסטרקתית היא כזאת שניתנת להורשה בלבד, שכן צריך ליישם אותה במחלקה שיורשת.

מחלקה זו נועדה להגדיר את הפעולות של כפתורים השייכים לפרויקט.

מחלקת UserKeys

מחלקה זו יורשת את מחלקת BaseKeys ואחראית על הכפתורים השייכיולשחקן במשחק.

המחלקה מממשת את הפעולות שהוגדרו מחלקת BaseKeys.

קוד –

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Linq;
using System.Collections.Generic;

namespace OrganizeProject
{
    public abstract class BaseKeys
    {
        public abstract Boolean RightPressed();
        public abstract Boolean LeftPressed();
        public abstract Boolean UpPressed();
        public abstract Boolean DownPressed();
        public abstract Boolean RightReleased();
        public abstract Boolean LeftReleased();
        public abstract Boolean UpReleased();
        public abstract Boolean DownReleased();
        public abstract Boolean ShiftPressed();
        public abstract Boolean SpacePressed();
    }

    public class UserKeys : BaseKeys
    {
        #region Data
        Keys right, left, up, down, run, fire;
        #endregion

        #region Ctors
        public UserKeys(Keys right, Keys left, Keys up, Keys down, Keys run,
Keys fire)
        {
            this.right = right;
```

```
            this.left = left;
            this.up = up;
            this.down = down;
            this.run = run;
            this.fire = fire;
        }
        #endregion

        public override Boolean RightPressed()
        {
            return Keyboard.GetState().IsKeyDown(right);
        }
        public override Boolean LeftPressed()
        {
            return Keyboard.GetState().IsKeyDown(left);
        }
        public override Boolean UpPressed()
        {
            return Keyboard.GetState().IsKeyDown(up);
        }
        public override Boolean DownPressed()
        {
            return Keyboard.GetState().IsKeyDown(down);
        }
        public override Boolean RightReleased()
        {
            return Keyboard.GetState().IsKeyUp(right);
        }
        public override Boolean LeftReleased()
        {
            return Keyboard.GetState().IsKeyUp(left);
        }
        public override Boolean UpReleased()
        {
            return Keyboard.GetState().IsKeyUp(up);
        }
        public override Boolean DownReleased()
        {
            return Keyboard.GetState().IsKeyUp(down);
        }
        public override Boolean ShiftPressed()
        {
            return Keyboard.GetState().IsKeyDown(run);
        }
        public override Boolean SpacePressed()
        {
            return Keyboard.GetState().IsKeyDown(fire);
        }
    }
}
```

מחלקת BotKeyBoard

מחלקה זו אחראית על הבינה המלאכותית של השחקן היריב. המטרה להפוך את השחקן היריב לכמה שיותר מתוחכם. פעילות הבוט מוגבלת ולא מזהה קופסאות נופלות, אך לה גם כמה יתרונות כגון ריצה מהירה באופן קבוע וקפיצה מעל כדור שחור ב 99% מהמקרים.

המטרה – לגרום לו לנצח אותי על מנת להפוך את המשחק למאתגר וכייפי!

מחלקה זו יורשת ממחלקת BaseKeys, על מנת לאפשר לשחקן לשחק.

פעולות –

- פעולת InitBot() המאפסת את תכונות הבוט ומאפשרת גישה לפעולת Update_Event().
- בנאי פשוט.
- פעולות גריסה לכל הפעולות האפשריות לשחקן. נחזיר true / false ובכך בעצם נשלוט בתזוזת השחקן. כל מה ש True הן פעולות שיבצע השחקן באותו רגע ו false לא.
- פעולת Update() אשר אחראית בין היתר על קפיצה לפני הגעת כדור שחור כדי להימנע מפגיעה בו. הוספתי לבוט זמן תגובה כדי למנוע תקיעות מצדו. הבוט בעצם זז לכדור הקרוב אליו ביותר ויורה כשהמרחק ביחס לציר X הוא בטווח שהוגדר. ברגע שכדור בורח הבוט ירוץ אליו כדי להשיג אותו ולירות.
- פעולת findClosestBall() המוצאת בעזרת חישובים מתמטיים את הכדור הקרוב ביותר. (כחול, וברגע שמופיע צהוב אז גם צהוב).

קוד -

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Collections.Generic;
using OrganizeProject.ProjectStates;


namespace OrganizeProject
{

    class BotKeyboard : BaseKeys
    {
        public GameObject bot;
        Vector2 direction;
        Ball target;
        float tagetDist;
        bool isShot;
        int shootDelay = 1500;
        int shootTime = 0;
        int findTargetTime = 0;
```

```
int findDelay = 5000;
bool isJump;


public void InitBot(GameObject bot)
{
    direction = Vector2.Zero;
    this.bot = bot;
    this.target = null;
    this.tagetDist = 0;
    isShot = false;

    GameState.Update_event += Update;
}

public BotKeyboard() : base()
{
}

public override Boolean RightPressed()
{
    return (direction.X > 0);
}
public override Boolean LeftPressed()
{
    return (direction.X < 0);
}
public override Boolean UpPressed()
{
    if (isJump)
    {
        isJump = false;
        return true;
    }
    return false;

}
public override Boolean DownPressed()
{

    return false;
}
public override Boolean RightReleased()
{
    return false;
}
public override Boolean LeftReleased()
{
    return false;
}
public override Boolean UpReleased()
{
    return false;
}
public override Boolean DownReleased()
{
    return false;
}
public override Boolean ShiftPressed()
{
    return true;
}
```

```
        public override Boolean SpacePressed()
        {
            return isShot;
        }
        public void Update()
        {
            if (GameState.isBlackBall)
            {
                if (S.blackBall.destinationRectangle.X < bot.Position.X &&
S.blackBall.velocity.X > 0)
                {
                    if (bot.Position.X - S.blackBall.destinationRectangle.X <=
350)
                        isJump = true;
                }
                if (S.blackBall.destinationRectangle.X > bot.Position.X &&
S.blackBall.velocity.X < 0)
                {
                    if (S.blackBall.destinationRectangle.X - bot.Position.X <=
350)
                        isJump = true;
                }
            }


            if (S.time - findTargetTime >= findDelay)
            {
                findClosestBall();
                findTargetTime = S.time;
            }
            if (isShot)
            {
                isShot = false;
                //shootTime = S.time;
            }
            else if (target != null)
            {
                if (bot.Position.X > target.destinationRectangle.X &&
target.velocity.X > 0)
                {
                    direction.X = -1;
                    if (tagetDist <= 500f)
                        direction.X = 1;

                    if (tagetDist <= 300f)
                    {
                        direction.X = 0;
                        direction.Y = 0;
                        isShot = true;
                        shootTime = S.time;
                    }
                }
                else if (bot.Position.X < target.destinationRectangle.X &&
target.velocity.X < 0)
                {
                    direction.X = 1;
                    if (tagetDist <= 500f)
                        direction.X = -1;

                    if (tagetDist <= 300f)
                    {
                        direction.X = 0;
```

```
                        direction.Y = 0;
                        isShot = true;
                        shootTime = S.time;
                    }
                }
                if (bot.Position.X < target.destinationRectangle.X &&
target.velocity.X > 0)
                {
                    direction.X = 1;
                }
                if (bot.Position.X > target.destinationRectangle.X &&
target.velocity.X < 0)
                {
                    direction.X = -1;
                }
            }
        }

        public void findClosestBall()
        {
            float minDist = float.MaxValue;
            float currDist;
            for (int i = 0; i < S.balls1.Count; i++)
            {

                currDist = Math.Abs(bot.Position.X -
S.balls1[i].destinationRectangle.X);

                if (minDist > currDist)
                {
                    minDist = currDist;
                    target = S.balls1[i];
                    tagetDist = minDist;
                }
            }

            for (int i = 0; i < S.pointBalls.Count; i++)
            {
                currDist = Math.Abs(bot.Position.X -
S.pointBalls[i].destinationRectangle.X);
                if (minDist > currDist)
                {
                    minDist = currDist;
                    target = S.pointBalls[i];
                    tagetDist = minDist;
                }
            }
        }





    }
}
```

מחלקת Camera

מחלקת Camera אחראית על מצלמת המשחק. כלומר, בפרויקט שלי אפשרתי לשחקן לבחור איך לראות את המסך, במלואו או חלקים ממנו.

אז יצרנו מחלקה שהמופע שלה מייצג מצלמה, המצלמה מקבלת בפעולה הבונה שלה מופע מסוג IFocus. ממשק זה משתמש במיקום הדמות ובכך מצייר את המסך בהתאם למיקומו.

על מנת להשתמש במצלמה נצטרך להיעזר במטריצות.
חשוב לציין שהפיקסלים שמצויירים הם אלו שזזים והמסך נשאר במקום.
באמצעות פונקציית CreateTranslation אנו לוקחים את הפיקסלים מה-Origin של המסך שהוא הקצה השמאלי העליון ומזיזים אותו אל הדמות. לאחר מכן, אנו מכפילים את החלק הראשון בפעולת מטריצה – CreateScale עם ה-scale שבחרנו ולבסוף על מנת שמרכז המצלמה יהיה אמצע המסך אנו מכפילים בעזרת פונקציית CreateTranslation עם הערכים של אמצע המסך.

כך למעשה אפשרנו לצייר רק חלק מן המסך על פי רצוננו ויצרנו אפקט של מצלמה שעוקבת בכל רגע.

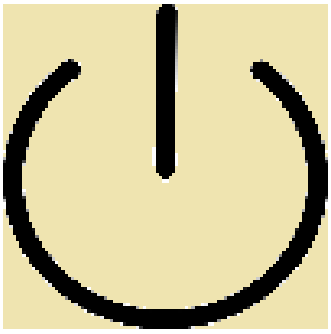
קוד -

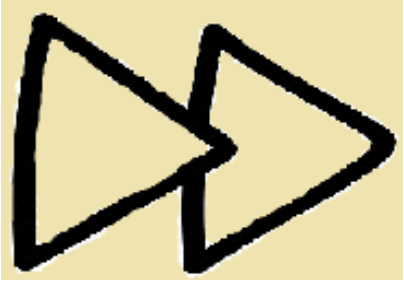
```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;

namespace OrganizeProject
{
    public class Camera
    {
        public Matrix Mat { get; private set; }
        public Matrix ScaleMatrix { get; private set; }
        MouseState currentMouse;
        MouseState previousMouse;

        int xTrans = S.screenWidth / 2;
        int yTrans = S.screenHeight / 2;

        int xCenter;
        int yCenter;

        float lerp = 0.93f;

        float scale;
        IFocus focus;
        Vector2 position;
        public Camera(IFocus focus)
        {
            this.focus = focus;
            position = Vector2.Zero;

            previousMouse = Mouse.GetState();
            scale = 1f;
        }

        public void Update()
```

```
        {
            xCenter = (int)(xTrans / scale);
            yCenter = (int)(yTrans / scale);

            currentMouse = Mouse.GetState();

            if (currentMouse.ScrollWheelValue - previousMouse.ScrollWheelValue
> 0)
            {
                scale += 0.1f;
                lerp = 1f;
            }
            else if (currentMouse.ScrollWheelValue -
previousMouse.ScrollWheelValue < 0)
            {
                scale -= 0.1f;
                lerp = 1f;
            }
            else
                lerp = 0.93f;

            if (scale < 0.4f)
                scale = 0.4f;

            ScaleMatrix = Matrix.Lerp(ScaleMatrix, Matrix.CreateScale(scale),
0.09f);

            previousMouse = currentMouse;


            //Mat = Matrix.CreateTranslation(new Vector3(-position, 0)) *
Matrix.CreateScale(0.8f) *
            //      Matrix.CreateTranslation(new Vector3(new Vector2(300, 480),
0));
            //position = Vector2.Lerp(focus.Position + Vector2.UnitY * 40f,
position, 0.93f);

            Mat = Matrix.CreateTranslation(new Vector3(-position, 0)) *
ScaleMatrix * Matrix.CreateRotationZ(0) *
                  Matrix.CreateTranslation(new Vector3(new Vector2(xTrans,
yTrans), 0));
            position = Vector2.Lerp(focus.Position + Vector2.UnitY * 40f,
position, lerp);


            if (position.X - xCenter < 0)
                position = new Vector2(xCenter, position.Y);
            if (position.Y - yCenter < 0)
                position = new Vector2(position.X, yCenter);

            if (position.X + xCenter > S.screenWidth*3)
                position = new Vector2(S.screenWidth*3 - xCenter, position.Y);
            if (position.Y + yCenter > (S.screenHeight- 30)*3)
                position = new Vector2(position.X, (S.screenHeight - 30) *3 -
yCenter);
        }
    }
                                                                            }
```

מחלקת Button

מחלקה זו אחראית על יצירת כפתור ומכילה את כל התכונות והפעולות הדרושות על מנת ליצור כפתור ולציירו על המסך.





פעולות -

- ציור – פעולה האחראית לצייר את הכפתור על המסך עם כל התכונות הנחוצות לציור.
- עדכון – עדכון הכפתור בהתאם למצב העכבר – האם לחוץ, האם העכבר מרחף מעל הכפתור וכו'.
- בנאים לסוגי כפתורים שונים.

קוד –

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Linq;
using System.Collections.Generic;
namespace OrganizeProject
{
    public class Button : Component
    {
        private MouseState currentMouse;

        private SpriteFont font;

        private bool isHovering;

        private MouseState previousMouse;

        private Texture2D texture;

        public event EventHandler Click;
```

```
        public bool Clicked { get; set; }

        public Color PenColor { get; set; }

        public Vector2 Position { get; set; }

        public float Scale { get; set; }

        public Color color { get; set; }

        public string Text { get; set; }

        public Rectangle Rectangle
        {
            get
            {
                return new Rectangle((int)Position.X - texture.Width / 2 *
(int)Scale, (int)Position.Y - texture.Height / 2 * (int)Scale, texture.Width *
(int)Scale, texture.Height * (int)Scale);
            }
        }


        public Button(Texture2D texture)
        {
            this.texture = texture;
            PenColor = Color.Black;
        }

        public Button(Texture2D texture, SpriteFont font)
        {
            this.texture = texture;
            this.font = font;
            PenColor = Color.Black;
        }

        public Button(Button button)
        {
            this.texture = button.texture;
            this.Position = button.Position;
            this.Scale = button.Scale;
            this.Text = button.Text;
            this.font = button.font;
            this.PenColor = button.PenColor;
        }

        public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
        {
              // Draws the rectangle
            spriteBatch.Draw(texture, Position, null, color, 0, new
Vector2(texture.Width / 2, texture.Height / 2), Scale, SpriteEffects.None, 0);

        }

        public override void Update(GameTime gameTime)
        {
            previousMouse = currentMouse;
            currentMouse = Mouse.GetState();

            var mouseRectangle = new Rectangle(currentMouse.X, currentMouse.Y,
1, 1);
```

```
            if (!Clicked)
            {
                isHovering = false;
                color = Color.White;

                if (mouseRectangle.Intersects(Rectangle))
                {
                    isHovering = true;
                    color = Color.Gray;

                    if (currentMouse.LeftButton == ButtonState.Released &&
previousMouse.LeftButton == ButtonState.Pressed)
                        Click?.Invoke(this, new EventArgs());
                }
            }
        }
    }
}
```

מחלקה אבסטרקטית State

מנגנון החלפת המסכים מנגנון החלפת המסכים משתמש במשתנים currentState ו – nextState שהוגדרו במחלקת Game1 על מנת לעקוב אחר מצב המשחק. בכל פעולת עדכון, המשתנה currentState מושווה למשתנה nextState כך שבתחילת העדכון הבא, הערך currentState ייצג את מצב המשחק נכון לעדכון הקודם. אם בעדכון מסוים יימצא ש – currentState שונה מ – nextState, המנגנון ידע שעליו לנקות את כל התוכן שעל המסך ולטעון מסך נוסף בהתאם לערך הנמצא ב- currentState . כאשר יש צורך לעבור ממסך אחד לאחר המחלקה שסיימה את תפקידה כמסך הנוכחי תודיע שיש לבצע החלפה על ידי גישה לGame1 ועדכון משתנה currentState למשתנה של המסך שיש לעבור אליו. מחלקה זו למעשה ממשת את מנגנון החלפת המסכים, כאשר היא מגדירה תכונות ופעולות אבסטרקטיות אשר יש לבצע במחלקות השייכות לכל מצב והן – MenuState ו – GameState.

- המחלקה וכל פעולותיה אבסטרקטיות על מנת לאפשר אך ורק למחלקה שיורשת לגשת שכן היא חייבת לממש את כל פעולות המחלקה האבסטרקטית.

פעולות-

- ציור – אחראית על ציור המצב הנתון.
- עדכון – עדכון המצב הנתון.
- אחרי עדכון – אחראית על שחרור משאבים שאינם נחוצים עוד.
- בנאי המאתחל את התכונות של המחלקה.

קוד -

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Linq;
using System.Collections.Generic;
using Microsoft.Xna.Framework.Content;

namespace OrganizeProject.ProjectStates
{
    public abstract class State
    {
        protected ContentManager content;

        protected GraphicsDevice graphicsDevice;

        protected Game1 game;

        public abstract void Draw(GameTime gameTime, SpriteBatch spriteBatch);

        public abstract void Update(GameTime gameTime);

        public abstract void PostUpdate(GameTime gameTime);

        public State(ContentManager content, GraphicsDevice graphicsDevice,
Game1 game)
        {
            this.content = content;
            this.graphicsDevice = graphicsDevice;
            this.game = game;
        }
    }
}
```

מחלקת MenuState

מחלקת MenuState מחלקה זו אחראית על מסך הפתיחה של המשחק ומציגה בפני השחקן אפשרויות לבחירת התחלת משחק או יציאה מהמשחק.

פעולות-

- בנאי MenuState - כאן נוצרים כל הכפתורים עבור התחלת משחק ויציאה ממנו. בנוסף קיימות פעולות המפעילות את כפתורים אלו.
- עדכון – עדכון הכפתורים בהתאם ללחיצות העכבר ומיקומו.
- ציור – ציור כל הרקע, הטקסטורות והכפתורים במסך הפתיחה.
- אחרי עדכון – אחראית על שחרור משאבים שאינם נחוצים עוד.

- מעבר על Components מאפשר מעבר על כל הכפתורים ועדכון והוספה שלהם לתוך רשימה.

```
components = new List<Component>()
            {
                newGameButton,
                quitGameButton
            };

foreach (var component in components)
                component.Draw(gameTime, spriteBatch);
```

קוד -

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Linq;
using System.Collections.Generic;
using Microsoft.Xna.Framework.Content;

namespace OrganizeProject.ProjectStates
{
    public class MenuState : State
    {

        private List<Component> components;


        public MenuState(Game1 game) : base(S.cm, S.gd, game)
        {
            var buttonTexture = game.Content.Load<Texture2D>("Goblin/Button");
            var ExitbuttonTexture =
game.Content.Load<Texture2D>("Goblin/ExitButton");
            var buttonFont = S.font;

            var newGameButton = new Button(buttonTexture)
            {
```

```
                Scale = 1f,
                Position = new Vector2(100, 590)

                //Text = "Start Game",
            };

            newGameButton.Click += NewGameButton_Click;

            var quitGameButton = new Button(ExitbuttonTexture)//, buttonFont)
            {
                Position = new Vector2(S.screenWidth / 2, 600),
                Scale = 1f
                //Text = "Quit Game",
            };

            quitGameButton.Click += QuitGameButton_Click;

            components = new List<Component>()
            {
                newGameButton,
                quitGameButton
            };


        }

        public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
        {
            spriteBatch.Begin();
            spriteBatch.Draw(game.Content.Load<Texture2D>("Goblin/Menu"), new
Vector2(0, 0), Color.White);


            foreach (var component in components)
                component.Draw(gameTime, spriteBatch);


            spriteBatch.End();
        }

        public void NewGameButton_Click(object sender, EventArgs e)
        {
            game.ChangeState(new GameState(game));
        }

        public override void PostUpdate(GameTime gameTime)
        {

        }

        public override void Update(GameTime gameTime)
        {
            foreach (var component in components)
                component.Update(gameTime);
        }

        public void QuitGameButton_Click(object sender, EventArgs e)
        {
            Console.WriteLine("Exit");
            game.Exit();
        }
    }
```

מחלקת GameState

מחלקה זו אחראית על כל הדברים הקשורים למסך המשחק ומהלך המשחק המתנהל בו.

על מנת לאפשר לכל אובייקט להתנהל בעצמו ללא תלות בקריאה חיצונית מ GameState יצרתי אירוע של עדכונים.

Delegate זה מצביע לפונקציה. Event של delegate זה כמו מערך של מצביעים לפונקציות מסוג מסוים. בסוף כשנרצה לקרוא לכל הפעולות יחדיו נקרא פשוט ל Event.
Event זה אירוע, הוא לא בהכרח מחזיק פונקציות. Event של delegate זה Eventשל פונקציות.

- Collision Event- אירוע זה בודק פגיעה של אובייקט בכדור.

```
public static event HandleCollision Collision_Event = null;

if (Collision_Event != null)
                {
                    Collision_Event(S.g);
                    Collision_Event(S.k);
                }
```

- Update Event- אירוע זה מעדכן את חבל השחקן ומגדיל אותו בגודל קבוע 60 פעמים בשנייה ברגע שמצויר.

```
public static event HandleUpdate Update_event = null;

if (Update_event != null)
                {
                    Update_event();
                }
```

- Fire Collision Event- אירוע זה בודק פגיעה של חבל הירייה בכדור. פגיעה של חבל לכדור מתאים תגרום להתפוצצות שלו לשני כדורים קטנים פי שתיים שינועו לשני כיווני המסך.

```
public static event HandleCollisionball FireCollision_Event = null;

if (FireCollision_Event != null)
            {
                for (int i = 0; i < S.balls.Count; i++)
                    FireCollision_Event(S.balls[i]);

                for (int i = 0; i < S.balls1.Count; i++)
                    FireCollision_Event(S.balls1[i]);

                for (int i = 0; i < S.pointBalls.Count; i++)
                    FireCollision_Event(S.pointBalls[i]);
            }
```

פעולות-

בנאי Gamestate - יצירת השחקנים והשמתם במפת המשחק. בנוסף יצירת רקע למשחק, עדכון כל הכדורים, עדכון הקרקע וגודלה עדכון השחקנים, בניית קופסאות, מצלמה וכו'.

עדכון – עדכון כל מצב המשחק הכולל – שחקנים, מפה ואובייקטים נוספים. נגריל מספר קופסה רנדומלי בעת נפילה. משחק יתחיל לאחר שלוש שניות בהם נראה רק את רקע המשחק. עדכון נפילת כדורים צהובים ויציאת כדורים שחורים רק החל מרגע מסוים לפי השעון. כמו כן הפעלת האירועים.

ציור – ציור כל הכדורים, הטקסטורות והאובייקטים על המסך. יש להדגיש, כל המצוייר מופיע ונעלם לפי תנועת המצלמה לפי מה שהוגדר.

```
spriteBatch.Begin(SpriteSortMode.Deferred, null, null, null, null,
null, cam.Mat);
```

נצייר את הניקוד החיים ומספר היריות ללא תלות במצלמה לנוחות המשתמש.

אחרי עדכון – אחראית על שחרור משאבים שאינם נחוצים עוד.

בדיקת ניצחון – פעולת EndGame() אשר מטרתה לבדוק מנצח במשחק ובהתאם להציג את מסך הסיום. משחק נגמר ברגע שנגמר הזמן או ברגע שהחיים של אחד השחקנים נגמרו.

קוד -

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Linq;
using System.Collections.Generic;
using Microsoft.Xna.Framework.Content;

namespace OrganizeProject.ProjectStates
{
    public class GameState : State
    {
        public static event HandleCollision Collision_Event = null;
        public static event HandleUpdate Update_event = null;
        public static event HandleCollisionball FireCollision_Event = null;
        public static event HandleCollisionBox BoxCollision_Event = null;
        public static event HandlePlayerCollision PlayerCollision_Event = null;

        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        public Drawable background;

        int lastBoxTime;
        int boxDelay = 10000;
        int startTime = S.time + 3000;
        int lastMinute = S.time + 60000;
        bool isYellowBalls = false;
        public static bool isBlackBall = false;
        public Camera cam;
        //Ball blackBall;
```

```
        public GameState(Game1 game) : base(S.cm, S.gd, game)
        {
            S.gameState = this;

            background = new
Drawable(game.Content.Load<Texture2D>("Goblin/BackGround"), new Vector2(0, 0),
null, Color.White, 0, Vector2.Zero, new Vector2(3f), SpriteEffects.None, 0);
            //TheDic.Init();
            S.supBox = new SupriseBox[7];

            S.balls = new List<Ball>();
            S.balls1 = new List<Ball>();
            S.pointBalls = new List<Ball>();

            S.Create_Ground_Line(3f);

            lastBoxTime = 0;


            S.g = new GameObject(Heros.Goblin, new Vector2(2500, 200),
Color.Red, 0, new Vector2(2f), 0, new UserKeys(Keys.Right, Keys.Left, Keys.Up,
Keys.Down, Keys.LeftShift, Keys.Space));
            S.k = new GameObject(Heros.Man, new Vector2(2000, 200), Color.Blue,
0, new Vector2(2f), 0, new BotKeyboard());
            BotKeyboard botKey = (BotKeyboard)S.k.keyboard;
            botKey.InitBot(S.k);
            cam = new Camera(S.g);

            S.balls.Add(new Ball(new Vector2(700, 20), Color.Red, new
Vector2(1f), 0, game.Content.Load<Texture2D>("Goblin/Ball2"), new
Rectangle(800, 20, 100, 100), 50, 50));
            S.balls.Add(new Ball(new Vector2(600, 20), Color.Red, new
Vector2(1f), 0, game.Content.Load<Texture2D>("Goblin/Ball2"), new
Rectangle(700, 20, 100, 100), 50, 50));
            S.balls.Add(new Ball(new Vector2(500, 20), Color.Red, new
Vector2(1f), 0, game.Content.Load<Texture2D>("Goblin/Ball2"), new
Rectangle(600, 20, 100, 100), 50, 50));
            S.balls.Add(new Ball(new Vector2(400, 20), Color.Red, new
Vector2(1f), 0, game.Content.Load<Texture2D>("Goblin/Ball2"), new
Rectangle(500, 20, 100, 100), 50, 50));

            S.balls1.Add(new Ball(new Vector2(300, 20), Color.Blue, new
Vector2(1f), 0, game.Content.Load<Texture2D>("Goblin/Ball2"), new
Rectangle(400, 20, 100, 100), 50, 50));
            S.balls1.Add(new Ball(new Vector2(200, 20), Color.Blue, new
Vector2(1f), 0, game.Content.Load<Texture2D>("Goblin/Ball2"), new
Rectangle(300, 20, 100, 100), 50, 50));
            S.balls1.Add(new Ball(new Vector2(100, 20), Color.Blue, new
Vector2(1f), 0, game.Content.Load<Texture2D>("Goblin/Ball2"), new
Rectangle(200, 20, 100, 100), 50, 50));
            S.balls1.Add(new Ball(new Vector2(50, 20), Color.Blue, new
Vector2(1f), 0, game.Content.Load<Texture2D>("Goblin/Ball2"), new
Rectangle(100, 20, 100, 100), 50, 50));


            S.supBox[0] = new SupriseBox(new Vector2(S.rnd.Next(0, (int)(850 *
S.scale)), 0), Color.White, new Vector2(0.4f), 0,
game.Content.Load<Texture2D>("SupriseBox/TwoArrows"), new Rectangle(200, 200,
40, 40), 0);
            S.supBox[1] = new SupriseBox(new Vector2(S.rnd.Next(0, (int)(850 *
S.scale)), 0), Color.White, new Vector2(0.4f), 0,
```

```
game.Content.Load<Texture2D>("SupriseBox/MoreHealth"), new Rectangle(200, 200,
40, 40), 1);
            S.supBox[2] = new SupriseBox(new Vector2(S.rnd.Next(0, (int)(850 *
S.scale)), 0), Color.White, new Vector2(0.4f), 0,
game.Content.Load<Texture2D>("SupriseBox/MorePoints"), new Rectangle(200, 200,
40, 40), 2);
            S.supBox[3] = new SupriseBox(new Vector2(S.rnd.Next(0, (int)(850 *
S.scale)), 0), Color.White, new Vector2(0.4f), 0,
game.Content.Load<Texture2D>("SupriseBox/MoreTime"), new Rectangle(200, 200,
40, 40), 3);
            S.supBox[4] = new SupriseBox(new Vector2(S.rnd.Next(0, (int)(850 *
S.scale)), 0), Color.White, new Vector2(0.4f), 0,
game.Content.Load<Texture2D>("SupriseBox/Freeze"), new Rectangle(200, 200, 40,
40), 4);
            S.supBox[5] = new SupriseBox(new Vector2(S.rnd.Next(0, (int)(850 *
S.scale)), 0), Color.White, new Vector2(0.4f), 0,
game.Content.Load<Texture2D>("SupriseBox/LessHealth"), new Rectangle(200, 200,
40, 40), 5);
            S.supBox[6] = new SupriseBox(new Vector2(S.rnd.Next(0, (int)(850 *
S.scale)), 0), Color.White, new Vector2(0.4f), 0,
game.Content.Load<Texture2D>("SupriseBox/LessPoints"), new Rectangle(200, 200,
40, 40), 6);


            S.supBox[0].isDraw = false;
            S.supBox[1].isDraw = false;
            S.supBox[2].isDraw = false;
            S.supBox[3].isDraw = false;
            S.supBox[4].isDraw = false;
            S.supBox[5].isDraw = false;
            S.supBox[6].isDraw = false;
        }



        public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
        {
            spriteBatch.Begin(SpriteSortMode.Deferred, null, null, null, null,
null, cam.Mat);
            background.Draw();
            if (S.time >= 3000)
            {
                S.g.UpdateState();
                if (S.balls.Count != 0)
                    for (int i = 0; i < S.balls.Count; i++)
                    {
                        S.balls[i].UpdateBall();
                    }

                S.k.UpdateState();
                if (S.balls1.Count != 0)
                    for (int i = 0; i < S.balls1.Count; i++)
                    {
                        S.balls1[i].UpdateBall();
                    }

                if (S.pointBalls.Count != 0)
                    for (int i = 0; i < S.pointBalls.Count; i++)
                    {
                        S.pointBalls[i].UpdateBall();
                    }
```

```
                if (isBlackBall)
                {
                    S.blackBall.UpdateBall();
                }
            }

            S.supBox[0].Draw();
            S.supBox[1].Draw();
            S.supBox[2].Draw();
            S.supBox[3].Draw();
            S.supBox[4].Draw();
            S.supBox[5].Draw();
            S.supBox[6].Draw();



            spriteBatch.End();
            spriteBatch.Begin();
            spriteBatch.DrawString(S.font, "Health: " + S.g.count1, new
Vector2(900, 5), Color.Brown);
            spriteBatch.DrawString(S.font, "Points: " + S.g.countPts, new
Vector2(900, 25), Color.Brown);
            spriteBatch.DrawString(S.font, "Max Shots: " + S.g.maxFireCount,
new Vector2(900, 45), Color.Brown);
            if (S.time > 3000)
                spriteBatch.DrawString(S.font, "" + (S.gameCDTimer - S.time) /
1000, new Vector2(480, 5), Color.Black);
            else
            {
                if (S.gameCDTimer - S.time >= 0)
                    spriteBatch.DrawString(S.font, "" + (startTime - S.time) /
1000, new Vector2(480, 15), Color.Black);

            }


            spriteBatch.DrawString(S.font, "Max Shots: " + S.k.maxFireCount,
new Vector2(10, 45), Color.Blue);
            spriteBatch.DrawString(S.font, "Health: " + S.k.count1, new
Vector2(10, 5), Color.Blue);
            spriteBatch.DrawString(S.font, "Points: " + S.k.countPts, new
Vector2(10, 25), Color.Blue);
        }

        public void EndOfGame()
        {
            if (S.gameCDTimer - S.time <= 0)
            {
                if (S.g.countPts > S.k.countPts)
                    game.ChangeState(new EndState(game, this, 1));
                else if (S.g.countPts < S.k.countPts)
                    game.ChangeState(new EndState(game, this, 2));
                else
                    game.ChangeState(new EndState(game, this, 0));
            }

            if(S.g.count1 == 0)
                game.ChangeState(new EndState(game, this, 2));
           if (S.k.count1 == 0)
                game.ChangeState(new EndState(game, this, 1));
        }
```

```
        public override void PostUpdate(GameTime gameTime)
        {
        }

        public override void Update(GameTime gameTime)
        {
            cam.Update();
            S.time = (int)gameTime.TotalGameTime.TotalMilliseconds;

            if (S.time >= 3000)
            {
                if (S.time - lastBoxTime > boxDelay)
                {
                    S.i = S.rnd.Next(0, 7);
                    S.supBox[S.i].Position = new Vector2(S.rnd.Next(0,
(int)(850 * S.scale)), 0);
                    S.supBox[S.i].isDraw = true;
                    lastBoxTime = S.time;

                }
                S.supBox[S.i].UpdateBox();
            }

            if ((S.gameCDTimer - S.time) / 1000 == 100 && isYellowBalls ==
false)
            {
                isYellowBalls = true;
                S.pointBalls.Add(new Ball(new Vector2(350, 20), Color.Yellow,
new Vector2(1f), 0, game.Content.Load<Texture2D>("Goblin/Ball2"), new
Rectangle(350, 20, 100, 100), new Vector2(-7f, -12f), S.rnd.Next(1, 100),
S.rnd.Next(1, 100)));
                S.pointBalls.Add(new Ball(new Vector2(450, 20), Color.Yellow,
new Vector2(1f), 0, game.Content.Load<Texture2D>("Goblin/Ball2"), new
Rectangle(550, 20, 100, 100), new Vector2(-7f, -12f), S.rnd.Next(1, 100),
S.rnd.Next(1, 100)));
            }



            if (((S.gameCDTimer - S.time) / 1000) %30 == 0 && isBlackBall ==
false)
            {
                isBlackBall = true;
                S.blackBall = new Ball(new Vector2(0, S.ground[0]-40),
Color.Black, new Vector2(1f), 0, game.Content.Load<Texture2D>("Goblin/Ball2"),
new Rectangle(0, S.ground[0]-40, 30, 30), new Vector2(20f, 0f), 0, 50);
            }
            if (isBlackBall && S.blackBall.countHitsWall >= 2)
            {
                S.blackBall.countHitsWall = 0;
                S.blackBall.isDraw = false;
                isBlackBall = false;
            }



            if (PlayerCollision_Event != null)
            {
                PlayerCollision_Event(S.g);
                PlayerCollision_Event(S.k);
            }
```

```
            if (BoxCollision_Event != null)
            {
                BoxCollision_Event(S.g);
                BoxCollision_Event(S.k);
            }


            if (Collision_Event != null)
            {
                Collision_Event(S.g);
                Collision_Event(S.k);
            }

            if (Update_event != null)
            {
                Update_event();
            }

            if (FireCollision_Event != null)
            {
                for (int i = 0; i < S.balls.Count; i++)
                    FireCollision_Event(S.balls[i]);

                for (int i = 0; i < S.balls1.Count; i++)
                    FireCollision_Event(S.balls1[i]);

                for (int i = 0; i < S.pointBalls.Count; i++)
                    FireCollision_Event(S.pointBalls[i]);
            }

            EndOfGame();
        }



    }
}
```

מחלקת EndState

מחלקה זו אחראית על מסך סיום המשחק.

ברגע שנגמר הזמן או לאחד השחקנים נגמרו החיים יש לעבור למסך הסיום. מסך זה יציג בפניי המשתמש האם ניצח הפסיד או המשחק נגמר בתיקו.

פעולות-

- בנאי EndState – כאן קיימת הגדרת משתנה בוליאני כ True אם המשחק נגמר בתיקו. בנוסף הגדרת תמונות של כפתורים וזימון פעולות להפעלת הכפתורים.
- עדכון – עדכון הכפתורים בהתאם ללחיצות העכבר ומיקומו.
- ציור – ציור הרקע במסך הסיום ועליו את הודעת תוצאת המשחק.
- אחרי עדכון – אחראית על שחרור משאבים שאינם נחוצים עוד.

- מעבר על Components מאפשר מעבר על כל הכפתורים ועדכון והוספה שלהם לתוך רשימה.

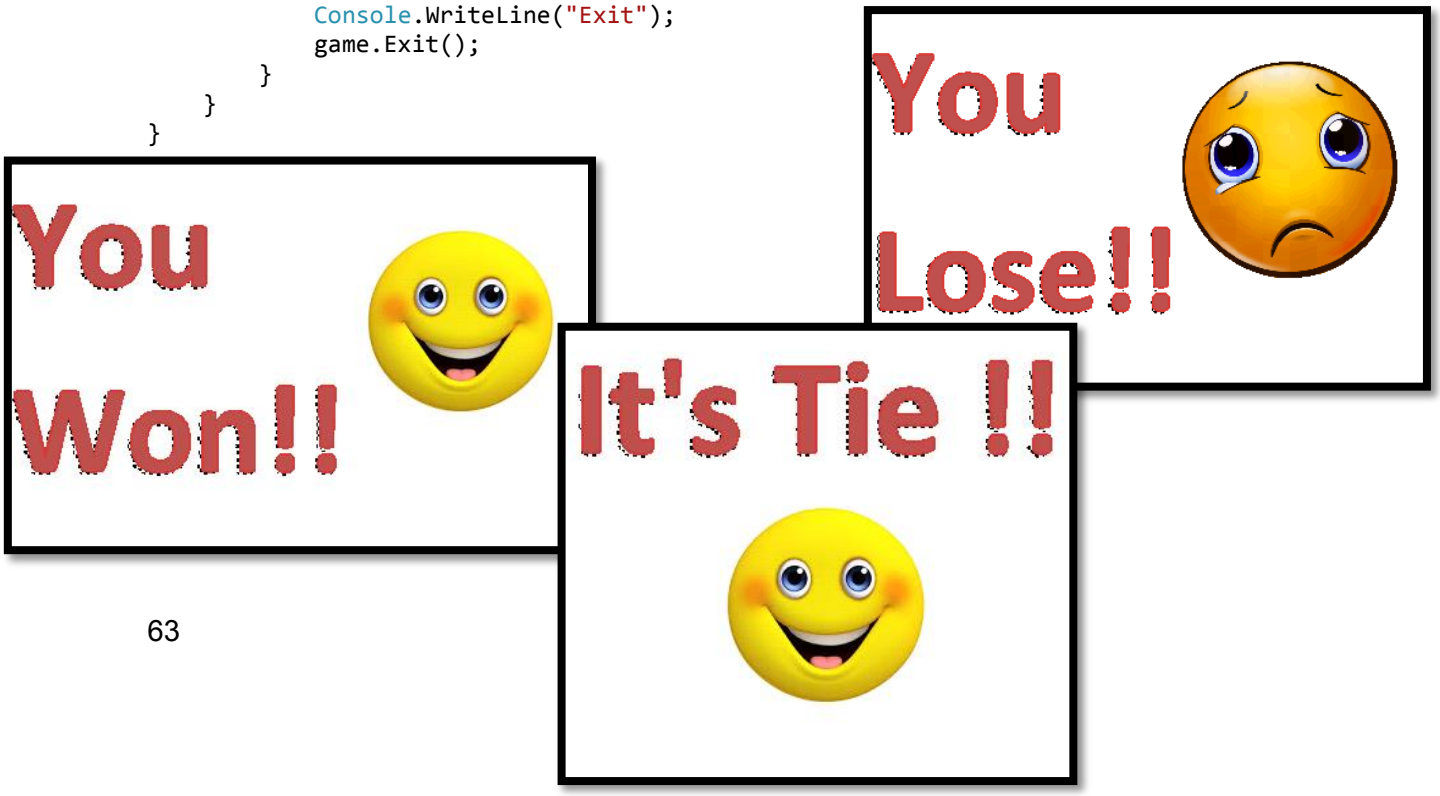
```
components = new List<Component>()
            {
                newGameButton,
                quitGameButton
            };

foreach (var component in components)
                component.Draw(gameTime, spriteBatch);
```

קוד –

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Linq;
using System.Collections.Generic;
using Microsoft.Xna.Framework.Content;

namespace OrganizeProject.ProjectStates
{
    public class EndState : State
    {
        private List<Component> components;
        private GameState gameState;
        private bool isTie = false;
        private int numOfTeamWon;

        public EndState(Game1 game, GameState gameState, int numOfTeamWon) :
base(S.cm, S.gd, game)
        {
            if (numOfTeamWon == 0)
                isTie = true;
```

```
            else
                this.numOfTeamWon = numOfTeamWon - 1;

            this.gameState = gameState;

            var buttonTexture = game.Content.Load<Texture2D>("Goblin/Button");
            var ExitbuttonTexture =
game.Content.Load<Texture2D>("Goblin/ExitButton");
            //var buttonTexture =
Dictionaries.TextureDictionary[FolderTextures.Controls]["Button"];
            var buttonFont = S.font;


            var newGameButton = new Button(buttonTexture)
            {
                Scale = 1f,
                Position = new Vector2(S.screenWidth / 2, 500)
                //Text = "Start Game",
            };

            newGameButton.Click += NewGameButton_Click;

            var quitGameButton = new Button(ExitbuttonTexture)
            {
                Position = new Vector2(S.screenWidth - 100, 600),
                Scale = 1f
                //Text = "Quit Game",
            };

            quitGameButton.Click += QuitGameButton_Click;

            components = new List<Component>()
            {
                newGameButton,
                quitGameButton
            };


        }

        public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
        {

            spriteBatch.End();
            spriteBatch.Begin(SpriteSortMode.Deferred, null, null, null, null,
null, gameState.cam.Mat);

            gameState.background.Draw();

            spriteBatch.End();
            spriteBatch.Begin();

            if (isTie)
            {
                Texture2D tieTexture =
game.Content.Load<Texture2D>("Goblin/Tie"); //
Dictionaries.TextureDictionary[FolderTextures.Graphics]["Tie"];
                spriteBatch.Draw(tieTexture, new Vector2(S.screenWidth / 2 -
tieTexture.Width / 2, 50), Color.White);
            }
            else
            {
```

```
                if (numOfTeamWon == 0)
                {
                    Texture2D wonTexture =
game.Content.Load<Texture2D>("Goblin/Win"); //
Dictionaries.TextureDictionary[FolderTextures.Graphics][General.teamColorString
[numOfTeamWon] + "Won"];
                    spriteBatch.Draw(wonTexture, new Vector2(S.screenWidth / 2
- wonTexture.Width / 2, 50), Color.White);
                }
                else
                {
                    Texture2D loseTexture =
game.Content.Load<Texture2D>("Goblin/Lose"); //
Dictionaries.TextureDictionary[FolderTextures.Graphics][General.teamColorString
[numOfTeamWon] + "Won"];
                    spriteBatch.Draw(loseTexture, new Vector2(S.screenWidth / 2
- loseTexture.Width / 2, 50), Color.White);
                }
            }
        }

        public void NewGameButton_Click(object sender, EventArgs e)
        {
            game.ChangeState(new MenuState(game));
        }

        public override void PostUpdate(GameTime gameTime)
        {

        }

        public override void Update(GameTime gameTime)
        {

            foreach (var component in components)
                component.Update(gameTime);


        }

        public void QuitGameButton_Click(object sender, EventArgs e)
        {
            Console.WriteLine("Exit");
            game.Exit();
        }
    }
}
```



רפלקציה

את הידע הראשוני שלי בפיתוח משחקים ב-xna רכשתי על ידי קריאה מדריכים באינטרנט והתנסות עצמאית, בעיקר באמצעות האחר www.riemers.net, לאחר שהרגשתי שצברתי מספיק ידע.

שיעורי ההדרכה לפרויקטים היו מלאי עניין והנאה והקלו עליי מאוד בעבודה על הפרויקט, תחילה למדנו על המבנה הבסיסי של המחלקות הראשיות עליו הומלצנו לבסס את הפרויקט, וכתבנו מחלקות-על לציור ולקלט מקלדת בפרויקט שבנינו לצורך הדגמה. בשלב הבא יצרנו מצלמת משחק באמצעות מתמטיקה וקטורים ומטריצות, השימוש בוקטורים ומטריצות ובאלגברה לינארית ככלל היה מאתגר בשבילי מכיוון שלא היה לי ידע נרחב בתחומים אלה קודם לכן, אף הרגשתי שלמידת נושאים אלה תרמה לי רבות.

מאוחר יותר עבדנו ביחד על פיתוח מנגנון לקריאה טקסטורות והפרדה ההמוניות ברצועות אנימציה על פי סימנים מוסכמים שקבענו מראש, אותם הוספנו לטקסטורוה הרצויות.

לבסוף, למדנו כיצד לנהל את פעולות האובייקטים המונפשים במשחק ולתמרן בין האנימציות המשתייכות להם על פי אופן פעולתם בכל רגע במהלך ריצה המשחק,

בחרתי לאמץ חלק מן הטכניקות ושיטות העבודה שנלמדו בכיתה ולפתח אותן בהתאם לצרכיי האישיים, אין לי ספק שהחלטה זו תרמה לי בפיתוח המשחק שלי וחסכה לי זמן עבודה יקר.

העבודה על הפרויקט הייתה אינטנסיבית וממושכת ודרשה מחויבות ומסירות עצומה, ועם זאת הייתה חווייתית, מלמדת ומאתגרת במובן הטוב, לצערי הרב, חלק מן הרעיונות הראשוניים שרציתי לשלב בפרויקט לא יצאו לפועל או מומשו חלקית בלבד עקב חוסר זמן, אף על פי כן, אני מרוצה מהתוצאות אליהן הגעתי ומהכישורים שרכשתי.

אסכם ואומר שנהניתי מאוד מביצוע מטלת הפרויקט מכיוון שהרגשתי חופשי להשתמש ביצירתיות ובמחשבה שלי, בין אם נאלצתי לעבוד עד השעות הקטנות של הלילה או לחפש אחר המשאבים הדרושים לי שעות וימים, לא הרגשתי שאני מבזבז את זמני אף לא לרגע.

ביבליוגרפיה:

1- s.netrwww.rieme - מדריכים לשימוש ב- XNA.
2- www.stackovefflow.com – בלוג המאפשר מתן עזרה ותמיכה בכל נושא הקשור לתכנות.
3- www.youtube.com – אשר אפשר לי ללמוד מסרטונים כיצד לבסס חלקים מהפרויקט.