Slugs

שם המפתח: איתן בקירוב

תעודת זהות: 209424811

שם המרצה: יהודה אור

תאריך הגשה: מאי 2018

מכללה: מכללת הכפר הירוק

תוכן עניינים

נושא הפרויקט

משחק דו ממדי בו משתתפות כמה קבוצות הנבחרות על ידי השחקן, אשר לכל אחת מהן מספר חיילים אשר נבחרים גם הם על ידי השחקן. מטרת כל קבוצה היא לחסל את כל קבוצות החיילים היריבות. כל קבוצה מקבלת ציוד בסיסי בעזרתו היא אמורה להשיג את המטרה – כגון בזוקה, תת-מקלע ושוטגאן. המשחק מתבצע בתורות כאשר בכל תור רק חייל אחד מתוך הקבוצה משחק ומטרתו לעשות כמה שיותר נזק לקבוצות האחרות או לפעול בצורה חכמה שתועיל לקבוצה בתורות הבאים. כאשר חייל נותר ללא חיים הוא מתפוצץ ואינו יכול להמשיך לשחק.

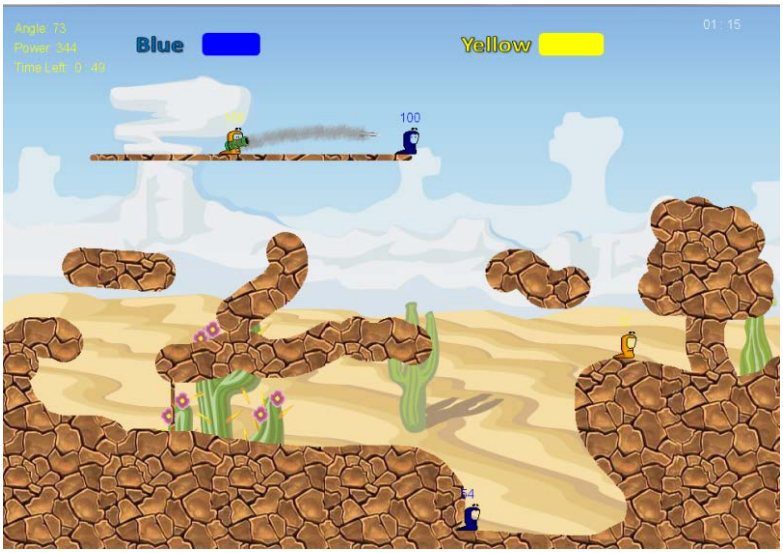
מפת המשחק בעלת מנוע הרס העובד על פי פיקסלים ההופך את מבנה המפה לדינאמי ומשתנה בהתאם לפעולות השחקן, כך לדוגמה, מפה שבתחילת המשחק הייתה בעלת הרים, גבעות ומנהרות, יכולה כולה להשתטח או אפילו כמעט ולהימחק דבר שיותיר לשחקנים מקום מינימלי לזוז בו.

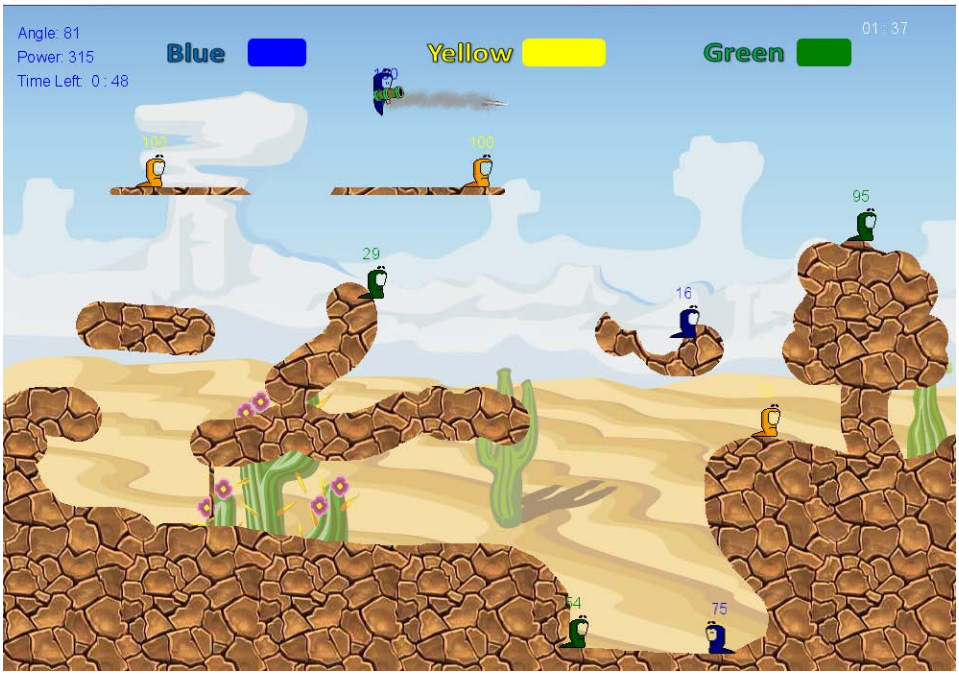
המשחק פותח בסביבת העבודה Visual Studio 2017 בשפת התכנות C# על ידי שימוש בספריית XNA 4.0 , תוספת חינמית ל-Visual Studio הכוללת ממשק עבודה נוח ואוסף נרחב של מחלקות וכלים מובנים המיועדים למפתחי משחקים.

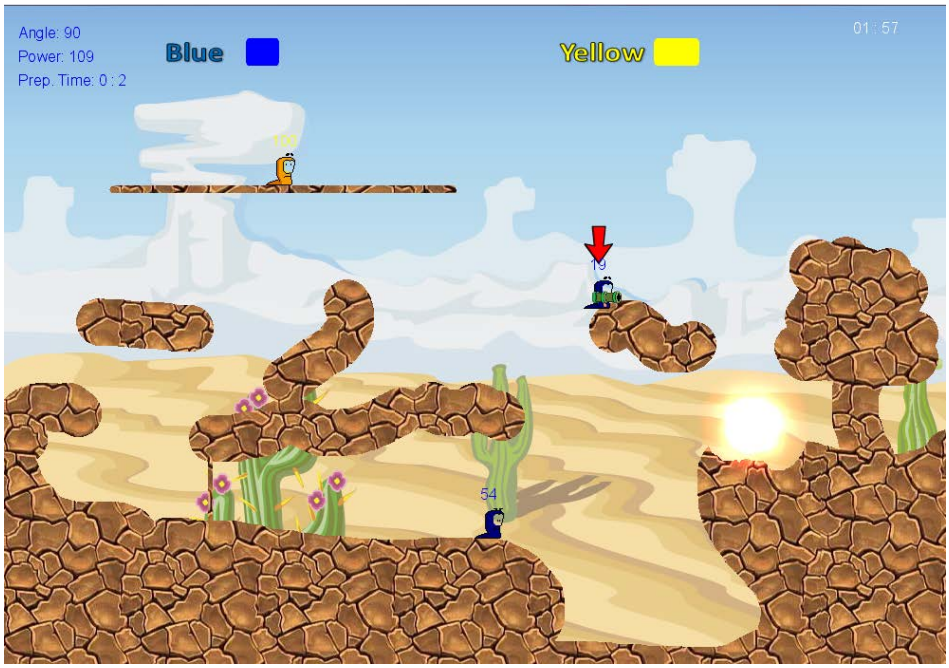
מטרת הפרויקט

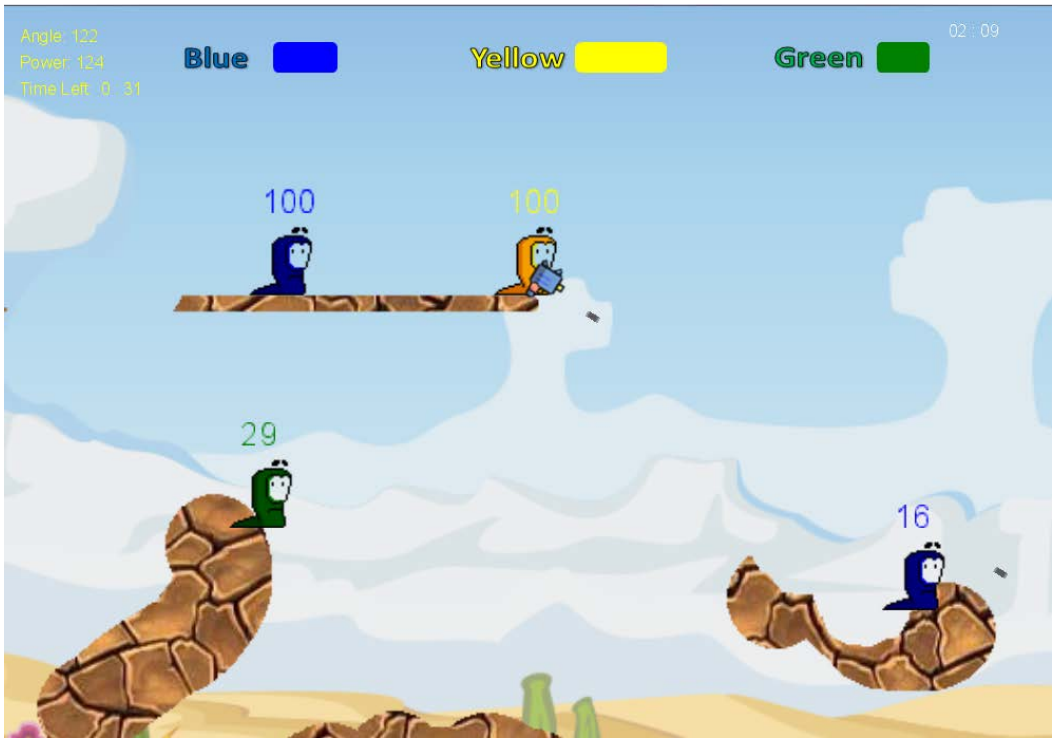
המטרה העיקרית של הפרויקט הייתה רכישת ניסיון בעבודה נרחבת בעלת ממדים גדולים, הדורaות שימוש שוטף בכישורים הנחוצים בתחום התכנות ובתכנות מונחה עצמים בפרט ומכסה נושאים ועקרונות תכנותיים רבים.
מטרות והשלכות משמעותיות נוספות של ביצוע מטלת הפרויקט היו הרחבת הידע והבקיאות האישית שלי בשפת C# ופיתוח הבנה עמוקה 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 הפרוייקט לשיפור קצב העבודה שלי ולצמצום זמן החשיבה הדרוש לי על מנת להגיע לתוצאות ולתוצרים ממשיים.


Slugs
Number of Teams:
2
3
4
Number of Players in a Team:
2
3
4
Start Game
Quit


Angle: 73
Power: 344
Time Left: 0 : 49
Blue
Yellow
01 : 15
100
100
54


Angle: 81
Power: 315
Time Left: 0 : 48
Blue
Yellow
Green
01 : 37
100
100
95
29
16
54
75


Angle: 90
Power: 109
Prep. Time: 0 : 2
Blue
Yellow
01 : 57
100
19
54


Angle: 122
Power: 124
Time Left: 0 : 31
Blue
Yellow
Green
02 : 09
100
100
29
16


Angle: 157
Power: 100
Blue
Yellow
Green
Red
00 : 53
Red
Won!!!
New Game
Quit

רפלקציה

את הידע הראשוני עולי 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 ומטריצוה 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- www.riemers.net - מדריכים לשימוש ב- XNA.
2- stackoverflow.com – בלוג המאפשר מתן עזרה ותמיכה בכל נושא הקשור לתכנות.
3- www.youtube.com – אשר אפשר לי ללמוד מסרטונים כיצד לבסס חלקים מהפרויקט.

מבוא ל #C

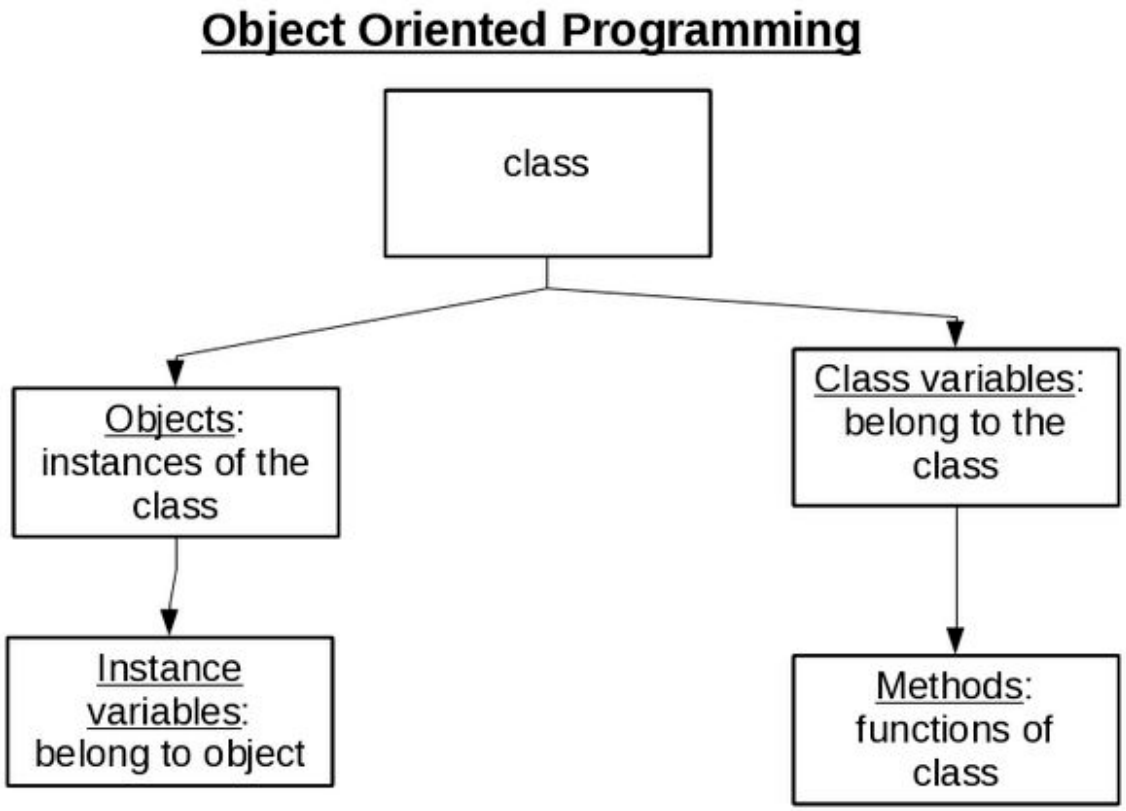
#C היא שפת תכנות עילית מרובת-פרדיגמות ומונחית עצמים שפותחה על ידי חברת מיקרוסופט בשנת 2000 כחלק מפרויקט דוט נט. השפה מאפשרת בנייה אלגנטית ונוחה של אפליקציות המשתמשות NET Frameworks. לרוב משתמשים בה לפיתוח אפליקציות למערכת ההפעלה Windows, שירותי אינטרנט של XML, תוכנות שרת-לקוח, מסדי נתונים, וכיוצא בזאת. ניתן גם, בין היתר, להשתמש בה לכתיבת קודים הפונים ל- Consoles,
כשפה מונחית עצמים, #C תומכת בעקרונות הכימוס, ההורשה ורב־הצורתיות. כל המשתנים והמתודות, כולל המתודה הראשית Main, כמוסות כהגדרות במחלקות-על נסתרות. #c מאפשרת לכל מחלקה לרשת באופן ישיר רק ממחלקה אחת, אך גם לממש מספר בלתי מוגבל של ממשקים.
קוד #C מהודר לשפת ביניים הנקראת (CIL (Common Intermediate Language. קוד זה מתקמפל לשפת מכונה בזמן ההרצה.

למה #C?

התחביר של שפת #C אקספרסיבי ומילולי ביותר, אך קל ופשוט ללמידה. השימוש בסוגריים מסולסלות לפתיחת וסגירת קטעי קוד וחוקים תחביריים בסיסיים נוספים יזוהו בקלות על ידי כל אדם בעל ניסיון מינימלי עם השפות C, C++ או Java, ולכן המעבר בין שפות אלה ל-c# מהיר וקצר.
השפה מפשטת הרבה מהמורכבויות של שפת C++, ומספקת כלים עוצמתיים כגון טיפוסים שניתן להזין בהם ערך null, אינומרציות(שימוש ב-enum), נציגים (delegates), שימוש ב- Properties לגישה לשדות כמוסים וכן גישה ישירה לזיכרון, אפשרות שאינה קיימת ב-Java.
בנוסף, #c משתמשת במנגנון ב- Garbage collector, כך שהמתכנת אינו צריך לדאוג לשחרור זיכרון של אובייקטים שאינם בשימוש כמו בשפות פרימיטיביות יותר.

תכנות מונחה עצמים

תכנות מונחה עצמים הוא פרדיגמת תכנות המשתמשת באובייקטים(עצמים) לשם תכנון תוכניות מחשבים ויישומים.
הפרדיגמה של תכנות מונחה עצמים צמחה מהתכנות הפרוצדורלי והמודולרי שבוסס על שגרות ופונקציות שפעלו על מבני נתונים חופשיים. שיטה זו יצרה 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 לצורך בתחזוקה רבה וכן איטיות ומורכבות בפיתוח ככל שהתקדם. כך, בניגוד למחירי החומרה שירדו בהתמדה, מחירי התוכנות דווקא עלו במהרה. לפיכך היה צורך לחפש פיתרון שיפשט ויקל על עבודת התכנות.
מרכיב בסיסי בתכנות מונחה-עצמים הוא המחלקה. מחלקה היא מבנה מופשט בעל תכונות המגדירות ומאפיינות את המחלקה כלפי חוץ, ופעולות, שהן פונקציות ייחודיות למחלקה. בחלק משפות התכנות קיימים גם אירועים, שהם שגרות השייכות למחלקה ומוזנקות בהקשרים שונים, למשל בתחילתו או בסיומו של הליך או כתגובה לקלט מהמשתמש.
על מנת להשתמש במחלקה, המתכנת יוצר מופעים שלה. כל מופע של מחלקה הוא עצם. היחס בין המחלקה לבין המופע, הוא היחס בין הגדרה של סוג טיפוס לבין הכרזה ויצירת משתנה מסוג זה בפועל בזיכרון המחשב. כלומר, בעוד שהמחלקה מהווה תבנית ומגדירה את התכונות והפעולות השייכות לה, עצם הוא משתנה שטיפוסו הוא המחלקה ומכיל את המידע הזה בפועל - כל עצם מכיל את התכונות והפעולות שהוגדרו במחלקה ממנה נוצר.
לדוגמה - למחלקה הקרויה אדם יש את התכונות: שם, מין, גיל, גובה ומשקל. כל אדם מסוים הוא עצם במחלקה זו, ולכל תכונה ערר המתאים לאותו העצם.



הורשה
אובייקטים המשתפים חלקים מהממשק שלהם, משתפים לעתים קרובות גם התנהגות. ירושה מאפשרת לממש את ההתנהגות המשותפת הזו 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 - כלומר להוסיף לה מימוש והתנהגות דרך מימושים של מתודות חדשות, הוספה של משתנים, ובמקרה הצורך ביצוע של דריסה של מתודות שהוגדרה במחלקה ממנה יורשים.
ניתן להגדיר 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, מחלקת אב או מחלקת־על.
המחלקה היורשת קרויה מחלקה נגזרת, בן, או תת־מחלקה.
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 ממחלקות הבסיס יורשות מאותה מחלקה. בשל כך חלק מהשפות, C# בפרט, מגבילות את המנגנון לירושה ממחלקת בסיס אחת, אך מאפשרות מימוש של מספר ממשקים.

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(down-casting).

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).

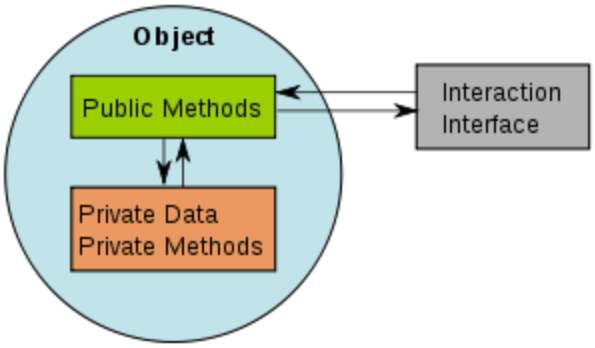
- כדי להריץ מתודות כאלה יש לבצע המרה למטה (down-casting) להפניה, כך שתהיה מאותו הטיפוס של האובייקט שבמחלקתו קיימת אותה פונקציה, ובכך להשיב לה הרשאות ולהריץ את הפונקציה.

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

מתודה היא רצף של פקודות המאוגדות יחדיו, במטרה לבצע מטלה מוגדרת, מימוש של אלגוריתם.

שגרות הן עיקרה של פרדיגמת התכנות הפרוצדורלי. מקובל כי שימוש מושכל בשגרות ובפונקציות עשוי לשפר את מבנה התוכנית, את קריאות הקוד ואת מידת הגמישות של התוכנית לביצוע שינויים. שימוש בשגרות 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 יכולות להיות מספר שגרות השייכות אליו. שגרה כזאת נקראת ־ "פונקציה חברה"(Member), והיא מגדירה את ההתנהגות של האובייקט עליו היא נקראה, ופועלות על המידע הכמוס בתוכו או בעזרתו. בהתאם למידת החשיפה שלה לשאר הקוד, שגרות אלה מהוות גם ממשק בין העצם לתוכנית כולה.

רקורסיה

רקורסיה היא פעולה הקוראת לעצמה כך שהיא יוצרת רצף של פעולות על עצמה עד לתנאי העצירה המפסיק את הקריאה.

רקורסיה הדדית מורכבת משתי פעולות כאשר פעולה א' קוראת לפעולה ב' ופעולה ב' קוראת לפעולה א', אם נביט על פעולה א' וב' כאחת נראה שהן מקיימות רקורסיה אחת רגילה.

אלגוריתם רקורסיבי הוא אלגוריתם אשר על מנת לפתור בעיה מסוימת, מפעיל את עצמו על מקרים פשוטים יותר של הבעיה. בדרך כלל יכלול האלגוריתם תנאי עצירה, שיביא להפסקת הרקורסיה ברמה שבה הפתרון נתון מראש, שאם לא כן תהיה זו לולאה אינסופית.

ממשק (Interface)

ממשק משמש לאבסארקציה של מחלקות התוכנה, ומגדיר את הפונקציות שעל מחלקה לממש כדי להיות שייכת אליו. במילים אחרות, אם אובייקט מממש ממשק כלשהו, אז הדבר מבטיח שלאובייקט תהיה התנהגות מסוימת. מימוש של ממשק הוא התחייבות של האובייקט למלא אחר מפרט של דרישות להתנהגות מסוימת, ולכן ניתן לראות בממשק כמעין חוזה. כאשר מחלקה מממשת את כל הסונקציות המוגדרות בממשק ניתן ליצור מופע שלה, אחרת היא נחשבת מימוש אבסטרקטי – כזה שדורש הרחבה על ידי מחלקה אחרת המשלימה את המימוש. שימוש בממשקים הוא נוהג של כתיבה נכונה בהנדסת תוכנה, כי בשיטה זו מתבצעת הפרדה בין המימוש בפועל לבין הדרישות שמאופיינות בממשק.

מחלקה אבסטרקטית (Abstract Class)

מחלקות אבסטרקטיות (מופשטות) הן מחלקות שמוגדרות לצורכי הורשה בלבד. מגדירים מחלקה אבסטרקטית כאשר קיים אותו מרכיב משותף לכמה מחלקות. מחלקה בין היתר מהווה מנגנון המבטיח שניסיון להקצות משתנה מסוג המחלקה תגרור שגיאת קומפילציה.

מחלקה אבסטרקטית נבדלת מממשקים בכך שהיא יכולה להכיל משתנים וקוד ולא רק הגדרות, ושאם יורשים ממחלקה אבסטרקטית לא ניתן לרשת מחלקות נוספות, כולל מחלקות אבסטרקטיות נוספות.

נציג (Delegate)

תפקידו של ה- delegate הוא ליצור אובייקט המצביע על פונקציה. במקום להפעיל את הפונקציה

בצורה הרגילה, נפעיל אותה דרך ה-delegate. היתרון של עבודה עם נציג הוא בהצבעה על פונקציה שאנו לא מכירים בזמן כתיבה של רכיב (מחלקה/ פונקציה וכד') מסוים, ומי שקבע איזו פונקציה תופעל הוא מי שמשתמש ברכיב.

Delegate יכול להצביע גם על מספר פונקציות. בפיתוח משחקי מחשב נעשה שימוש תדיר בתכונה זו, בעיקר כאשר רוצים לגרום לכך שמספר פעולות ממחלקות שונות יפעלו על פי סדר מסוים ללא צורך לארגן אותן ביחד במקום אחד בקוד. סדר הפעלתן יהיה למעשה הסדר שבו רשם אותן המתכנת למופע סטטי של ה-delegate.

אירוע (Event)

Event – הודעה שאובייקט שולח לאובייקטים אחרים.

Event הינו למעשה מקרה פרטי של delegate. הרעיון־ של event הינו להצביע על פונקציה והוא מספק מספר הגבלות אשר לא קיימות ב-delegate רגיל:

- ניתן להגדיר event רק כחבר מחלקה (member) ולא בתוך פונקציה או כפרמטר לפונקציה.
- ניתן להפעיל event רק באותה מחלקה בה הוא נכתב.
- ניתן להוסיף ולהוריד פונקציות events באמצעות האופרטורים =+, =- , אך לא ניתן לדרוס את מה ש'ש שס באמצעות =.

פיתוח ב- XNA

ספריית XNA היא תוסף חינמי מבוסם .NET Framework ל-Visual Studio המכיל אוסף של כלים המיועדים לפיתוח משחקי מחשב בנוחות וביעילות. משחקים הנוצרים באמצעות XNA נכתבים בשפת #C וניתן לייצר עבורם גרסאות מתאימות ל- Xbox, Windows Phone ו- Windows NT.

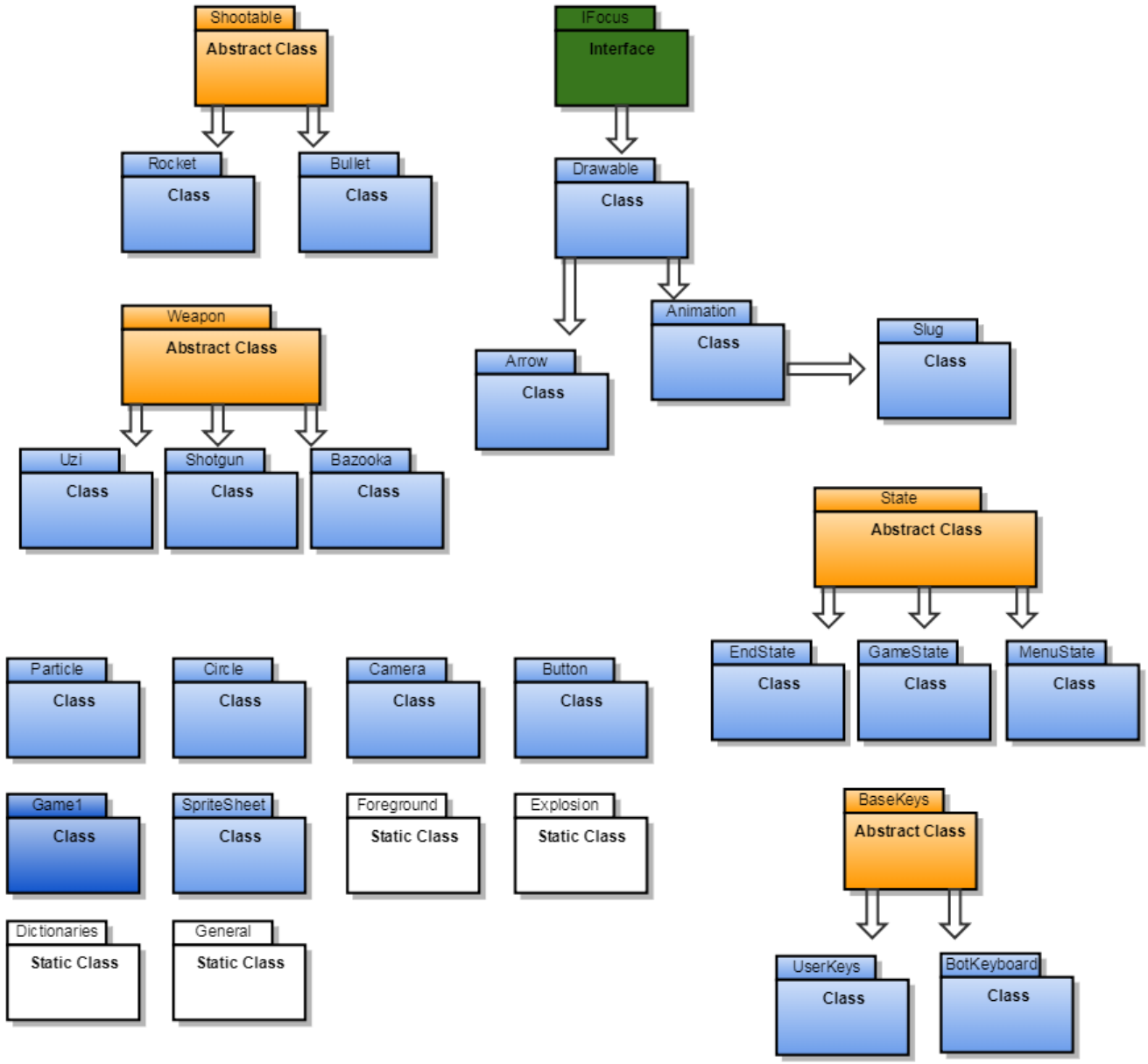
ספריית XNA הוכרזה לראשונה במרץ 2004 על ידי מיקרוסופט וגרסתה הראשונה שוחררה שנתיים מאוחר יותר. הגירסה העדכנית ביותר של הספרייה היא xna 4.0 /שיצאה בשנת 2010 והביאה איתה את אפשרות הפיתוח ל-Windows Phone 7.

במשחק Slugs, כמו במרבית משחקי הפלטפורמה, פועלים על העצמים המונפשים במשחק באופן מדומה כוחות פיזיקליים דו־מימדיים בסיסיים הדומים לכוחות החלים על עצמים ממשיים.

- כוח הכבידה - על פי ההגדרה, כוח הכבידה הוא כוח המושך בין גופים בהתאם למכפלת המסות שלהם ומרחקם אחד מהשני. כוח הכבידה הוא האחראי לכך שגופים ייפלו כלפי מטה ולא ירחפו באוויר (מסתם של גופים קטנים זניחה ביחס לכוכב שהם נמצאים עליו, ולכן השפעתם על תנועתו לא ניכרת).
כוח הכבידה של כוכב פועל תמיד אל עבר מרכז הכוכב, ולכן כאשר עצם כלשהו עומד במצב מנוחה על מישור, כיוון כוח המשיכה הפועל עליו ניצב לקרקע.
כוח הכבידה פועל על כל גוף בכל עת, בין אם הגוף במנוחה או בתנועה.
גופים הנמצאים באוויר ינועו בהשפעת כוח הכובד בלבד (כל עוד לא פועלים עליהם כוחות נוספים). לתופעה זו קוראים נפילה חופשית. תאוצת הנפילה החופשית איננה תלויה במסת הגוף הנופל, אלא רק בעוצמת שדה הכבידה. מעובדה זו נובע שמהירותו של גוף הנופל לכיוון פני כוכב תלויה בתאוצת הנפילה החופשית המקורבת של הכוכב ובזמן הנפילה של הגוף (כל עוד התנגדות האוויר אינה זניחה), כך שככל שהגוף נמצא בנפילה כלפי מטה יותר זמן, כך מהירותו יותר גדולה.

- הכוח הנורמלי - כוח שמפעיל משטח בתגובה לכוח שמופעל עליו. כיווט תמיד יהיה בניצב למשטח נגד כיוון הכוח המופעל על המשטח. כאשר הגוף עומד במצב מנוחה על משטח, ניתן להסיק שהכוח הטרמלי הוא כוח המנוגד בכיוונו ושווה בגודלו לכוח המשיכה הפועל על אותו הגוף מהסיבה שאם הדבר היה אחרת ושקול הכוחות בציר ה־ץ היה שונה מ־0, אותו הגוף היה טפל לתוך המשטח בתאוצת הכוכב, על פי החוק השני של ניוטון.

מפת UML


Shootable
Abstract Class
Rocket
Class
Bullet
Class
IFocus
Interface
Drawable
Class
Weapon
Abstract Class
Uzi
Class
Shotgun
Class
Bazooka
Class
Arrow
Class
Animation
Class
Slug
Class
State
Abstract Class
EndState
Class
GameState
Class
MenuState
Class
Particle
Class
Circle
Class
Camera
Class
Button
Class
Game1
Class
SpriteSheet
Class
Foreground
Static Class
Explosion
Static Class
Dictionaries
Static Class
General
Static Class
BaseKeys
Abstract Class
UserKeys
Class
BotKeyboard
Class

מחלקות המערכת

ממשק IFocus

ממשק זה יהיה שימושי למצלמה במשחק.

הממשק מחזיק רק במשתנה אחד – מיקום האובייקט על המסך.

הממשק הינו לנוחיות בלבד, כלומר, כל אובייקט שישלח למצלמה כדי שתתפקס עליו, יתקבל רק המיקום שלו כיוון שזה ירש את ממשק זה.

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

מחלקת Camera

מחלקת Camera אחראית על מצלמת המשחק. כלומר, בפרויקט שלי אפשרתי לשחקן לבחור איך לראות את המסך, במלואו או חלקים ממנו כאשר בכל רגע המצלמה מפוקסת על השחקן אשר תורו לשחק – עוקבת אחרי השחקן.

אז יצרנו מחלקה שהמופע שלה מייצג מצלמה, המצלמה מקבלת בפעולה הבונה שלה מופע מסוג IFocus. ממשק זה משתמש ק 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.

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

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

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

        int xCenter;
        int yCenter;
        float lerp = 0.93f;
        float scale;
        IFocus focus;
```

```
        public Vector2 position;
        public Camera(IFocus focus)
        {
            this.focus = focus;
            position = Vector2.Zero;
            previousMouse = Mouse.GetState();
            scale = 1f;
        }
        public void ChangeCamFocus(IFocus focus)
        {
            this.focus = focus;
        }
        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 < 1)
                scale = 1;

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

            Mat = Matrix.CreateTranslation(-position.X, -position.Y, 0) *
ScaleMatrix *
                  Matrix.CreateTranslation(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 > General.screenWidth)
                position = new Vector2(General.screenWidth - xCenter, position.Y);
            if (position.Y + yCenter > General.screenHeight)
                position = new Vector2(position.X, General.screenHeight - yCenter);
        }
    }
}
```

מחלקת Drawable

מחלקה זו היא מחלקת אב לכל טקסטורה או טקסט שניתן לציור על המסך, כגון דמויות, חצים, טילים וכו'. במחלקה קיימות כל התכונות הדרושות לשם ציור האובייקט על המסך. כאשר כל אובייקט משתמש בתכונות האופייניות לציור שלו ולכן קיימים בנאים שונים התומכים באפשרות זו.

הפעולה היחידה שקיימת במחלקה היא הפעולת Draw אשר אחראית לצייר את האובייקטים שיורשים אותה. חשוב לציין שאם אובייקט משתמש בציור עם תכונות השונות מתכונות ציור אלה, הוא עוקף את הפעולה במחלקה שלו ומשתמש בה לצורך הציור שלו על המסך.

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

namespace Slugs
{
    public class Drawable : IFocus
    {
        #region Data
        public Texture2D Texture { get; set; }
        public Rectangle? SourceRectangle { get; set; }
        public Rectangle DestionationRectangle { 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 float Scale { get; set; }
        public float LayerDepth { get; set; }

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

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

        public Drawable(Rectangle destionationRectangle, Color color)
        {
            DestionationRectangle = destionationRectangle;
            this.color = color;
        }

        public Drawable(Texture2D texture, Vector2 position, Rectangle?
sourceRectangle,
                        Color color, float rotation, Vector2 origin, float 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;
            this.LayerDepth = layerDepth;
        }
        public Drawable(Color color, float rotation, float scale, float layerDepth)
        {
            this.color = color;
            this.Rotation = rotation;
            this.Scale = scale;
            this.LayerDepth = layerDepth;
        }

        public Drawable(Texture2D texture, Rectangle? sourceRectangle, Color color)
        {
            Texture = texture;
            SourceRectangle = sourceRectangle;
            this.color = color;
        }

        public Drawable(Texture2D texture)
        {
            Texture = texture;
        }

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

        public Drawable()
        {
        }
        public Drawable(Texture2D texture, Vector2 position, Color color, Vector2
origin)
        {
            Texture = texture;
            Origin = origin;
            Position = position;
            this.color = color;
        }
        #endregion

        public virtual void Draw()
        {
            General.sb.Draw(
                    Texture, Position, SourceRectangle,
                    color, Rotation, Origin,
                    Scale, Effects, LayerDepth);
        }
    }
}
```

המחלקה General

מחלקה זו הינה מחלקת עזר אשר מכילה פעולות משתנים ואובייקטים סטטיים שנעשה בהם שימוש גלובלי ותדיר בפרויקט כולו. כל הנתונים במחלקה זו נגישים מכל מחלקה בפרויקט וניתן לשנות אותם בהתאם לצורך.

כמו כן, מחוץ למחלקה (באזור ה-namespace), מוגדרים enums שנעשה בהם שימוש נרחב בפרויקט. תפקידם להפוך את הקוד לברור ומילולי יותר ובכך להקל על כתיבת והבנת הקוד.

ה-enums בפרויקט זה:
Hero – אשר מונה את סוגי השחקנים במשחק
PlayerState – אשר מונה את מצבי השחקן השונים, כגון, עמידה, הליכה וכו'
Tempo – אשר אחראי על תדירויות החלפת האנימציות לכל שחקן, לדוגמה, החלפת האנימציות במצב הליכה תהיה איטית יותר מזו של ריצה.

פעולות :

- NextPlayer – פעולה זו נועדה לטפל בהחלפת התורות בין השחקנים.
- GenerateTerrainMask – פעולה זו אחראית על מעבר על מסכת האדמה ושמיר במערך דו ממדי את הפיקסלים השחורים כאדמה והלבנים כאוויר.
- TextureTo2DArray – פעולה זו מקבלת טקסטורה ומחזירה מערך דו ממדי של צבעי הטקסטורה (קיימת פעולה מובנית אשר מעבירה למערך חד ממדי, אך פעולה זו מקלה על העבודה בהמשך).

```
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;

namespace Slugs
{
    public interface IFocus
    {
        Vector2 Position { get; }
    }
    public enum Heroes { Slug }
    public enum PlayerState { Stance, Run, Walk, Jump }
    public enum Tempo { Slow = 7, Medium = 5, High = 2 }
    public enum FolderTextures { Graphics, Controls }

    static class General
    {
        #region Data
        public static SpriteBatch sb;
        public static GraphicsDeviceManager gp;
        public static ContentManager cm;
        public static GraphicsDevice device;
        public static Game1 game;

        public static Camera cam;
        public static int Time = 0;
```

```
        public static SpriteFont font;

        public static int screenWidth;
        public static int screenHeight;
        public static Slug[] players;
        public static int numberOfPlayers;
        public static int currentPlayer = 0;
        public static int[] currentTeamHealth;
        public static string[] teamColorString = { "Blue", "Yellow", "Green", "Red"
};

        public static Random randomizer = new Random();
        public static int[,] terrainMask;
        #endregion

        public static void NextPlayer()
        {
            General.players[General.currentPlayer].IsPlaying = false;
            Slug previousSlug = General.players[General.currentPlayer];

            General.currentPlayer = General.currentPlayer + 1;
            General.currentPlayer = General.currentPlayer % numberOfPlayers;
            while (previousSlug != General.players[General.currentPlayer] &&
!General.players[General.currentPlayer].IsAlive)
                General.currentPlayer = ++General.currentPlayer % numberOfPlayers;

            cam.ChangeCamFocus(General.players[General.currentPlayer]);
            General.players[General.currentPlayer].StartPlaying();
        }
        public static void GenerateTerrainMask()
        {
            terrainMask = new int[screenWidth, screenHeight];
            Color[,] terrainColorArray =
TextureTo2DArray(Dictionaries.TextureDictionary[FolderTextures.Graphics]["terrain3"]
);
            for (int x = 0; x < screenWidth; x++)
            {
                for (int y = 0; y < screenHeight; y++)
                {
                    if (terrainColorArray[x, y] == Color.Black)
                        terrainMask[x, y] = 1;
                    else
                        terrainMask[x, y] = 0;
                }

            }
        }
        public static Color[,] TextureTo2DArray(Texture2D texture)
        {
            Color[] colors1D = new Color[texture.Width * texture.Height];
            texture.GetData(colors1D);

            Color[,] colors2D = new Color[texture.Width, texture.Height];
            for (int x = 0; x < texture.Width; x++)
                for (int y = 0; y < texture.Height; y++)
                    colors2D[x, y] = colors1D[x + y * texture.Width];

            return colors2D;
        }
    }
}
```

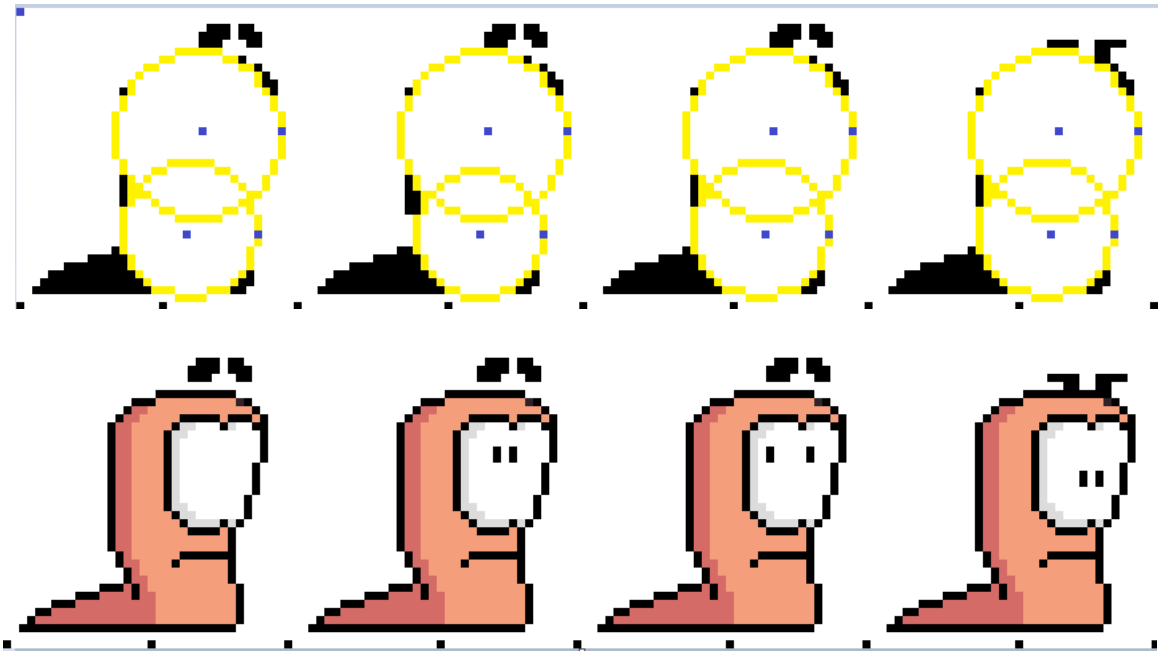
מנגנוניי ההתנגשות במשחק:

בפרויקט זה השתמשתי בשני מנגנוני התנגשות שונים למקרים שונים.

- הראשון עוסק בפגיעה של אובייקטים בשחקן.
- השני עוסק בהתנגשות החלקיקים באדמת המשחק.

המנגנון הראשון
על מנת לייעל את בדיקות פגיעת היריות בשחקן ולא לפעול לפי בדיקת פיקסלים בכל אחת מהטקסטורות השתמשתי בהגדרת עיגולים עבור כל טקסטורה אצל כל פריים בכל מצב שיש לשחקן.
כך בכל רגע נבדוק האם הירייה נמצאת בתוך רדיוסי העיגולים של השחקן ואם כן נפעל בהתאם.

כדי למצוא את העיגולים יצרתי עותק של הטקסטורה כאשר כל פריים שם בשחור לבן ושם הנחתי שתי נקודות, כאשר הראשונה הינה מרכז המעגל והשנייה למעשה יוצרת את הרדיוס עם הנקודה הראשונה. כך עשיתי כמספר פעמים עד שכל המעגלים כיסו כל פריים
ציירתי את הפיקסל הראשון בתמונה בצבע השונה משחור, לבן או צבע המעגל שציירתי וזאת על מנת שאוכל לדעת את הצבע אותו אני מחפש בתכנית כשאעבור על התמונה.



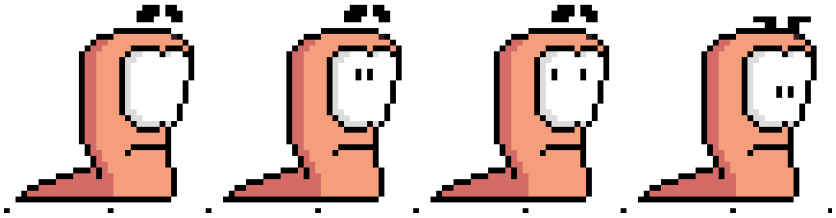
(על המנגנון השני אפרט בהמשך)

מנגנון יצירת האנימציה:

אנימציה היא תחום שבו יוצרים אשליה של תנועה על גבי מסך על ידי מספר תמונות (פריימים) בקצב מהיר המייצגות מצב כלשהו, למשל, הליכה, ריצה, קפיצה ועוד.

בפרויקט שלי על מנת ליצור אנימציה עבור הדמויות במשחק השתמשתי במנגנון אנימציה אשר יהודה אור לימד אותנו במהלך השנה ופועל באופן הבא:

- ניקח תמונה המתארת מצב כלשהו של דמות.
- נפרק את הפריימים על ידי ציור נקודות בתחתית התמונה בין כל פריים ובנוסף נצייר את נקודת המוצא שלפיה נוכל נצייר את הפריים על המסך.
- נעבור על כל תמונה ונשמור ברשימה אחת את כל הפריימים וברשימה אחרת את כל נקודות המוצא.
- ולבסוף נמחק את כל הרקע סביב הפריימים.



לכל דמות במשחק קיימים כמה מצבים ולכן נשמור בתוך מילון של הדמויות את כל המצבים של האנימציה הנתונים להם על פי המנגנון שהוצג לעיל.

מחלקת SpriteSheet

מחלקה זו למעשה מיישמת את מנגנון יצירת האנימציה.

פעולות:

- בנאי SpriteSheet כאשר בתוכו מתבצע כל אלגוריתם האנימציה עבור מצב מסויים של שחקן מסויים.
- Find Tempo - מציאת קצב מהירות החלפת הפריימים במצב מסויים.
- MakeTransparent - מחיקת הרקע של אותו מצב, כלומר, הפיכת הרקע לשקוף.
- CreateCircles – שמירת העיגולים של השחקן על פי המנגנון שהוצג לעיל.

```
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 Slugs
{

    public class SpriteSheet
    {
        #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; }

        public List<List<Circle>> AllPagesCircle { get; private set; }
        #endregion

        #region Ctors
        public SpriteSheet(Heroes hero, PlayerState state)
        {
            Recs = new List<Rectangle>();
            Orgs = new List<Vector2>();
            Tex = General.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));

            }
            for (int i = 0; i < pos.Count - 2; i += 2)
            {
                CreateCircles(hero, state);

            }
            Pace = FindTempo(state);
            // Make background color transparent
            MakeTransparent();
        }
        #endregion
        void CreateCircles(Heroes hero, PlayerState pState)
        {
            Texture2D maskTexture;
            Color[] c;
            int placeInRec;
            bool foundFirst;
            Color firstC;
            Vector2 placeCenter;
            float radius;

            if (File.Exists(Directory.GetCurrentDirectory() + "/Content/" + hero +
"/" + pState + "Mask" + ".xnb"))
            {
                maskTexture = General.cm.Load<Texture2D>(hero + "/" + pState +
"Mask");
                AllPagesCircle = new List<List<Circle>>();
                c = new Color[maskTexture.Width * maskTexture.Height];
                maskTexture.GetData<Color>(c);
                foundFirst = false;
                firstC = new Color();
```

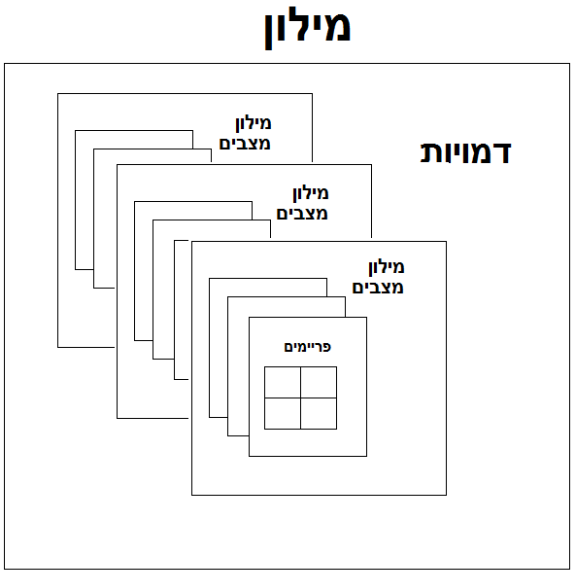
```
                placeCenter = new Vector2();
                radius = 0;
                for (int i = 0; i < Recs.Count; i++)
                {
                    AllPagesCircle.Add(new List<Circle>());
                    for (int j = 0; j < Recs[i].Height * Recs[i].Width; j++)
                    {
                        placeInRec = j % Recs[i].Width + Recs[i].X + j /
Recs[i].Width * maskTexture.Width;
                        if (i == 0 && j == 0)
                            firstC = c[0];
                        else if (c[placeInRec] == firstC && foundFirst == false)
                        {
                            foundFirst = true;
                            placeCenter = new Vector2(placeInRec %
maskTexture.Width, placeInRec / maskTexture.Width) - new Vector2(Recs[i].X +
Orgs[i].X, Orgs[i].Y);
                        }
                        else if(foundFirst == true && c[placeInRec] == firstC)
                        {
                            foundFirst = false;
                            radius = (new Vector2(placeInRec % maskTexture.Width,
placeInRec / maskTexture.Width) - new Vector2(Recs[i].X + Orgs[i].X, Orgs[i].Y) -
placeCenter).Length();
                            AllPagesCircle[i].Add(new Circle(placeCenter, radius));
                        }
                    }
                }
            }
        }
        private Tempo FindTempo(PlayerState state)
        {
            switch (state)
            {
                case PlayerState.Stance:
                    return Tempo.Slow;
                case PlayerState.Walk:
                    return Tempo.Slow;
                case PlayerState.Run:
                    return Tempo.Medium;
                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);
        }
    }
}
```

המחלקה הסטטית Dictionaries

מחלקת Dictionaries הינה מחלקה סטטית אשר מכילה בתוכה את כל המילונים שבפרויקט זה. כאן אנו יוצרים את מילון האנימציות PlayerAnimationDictionary עבור כל דמות אשר מחולק באופן הבא:

- קיימות כמה דמויות
- לכל דמות כמה מצבים
- כל מצב מורכב מכמה פריימים אשר מנפישים את האובייקט

מכאן נובע שכל מילון מורכב מכמה דמויות אשר להן קיים מילון של מצב וכל הפריימים השייכים לו:



במחלקה זו קיימת פעולת אתחול המילון אשר אחראית לעבור על כל הדמויות ולארגן את כל המידע הדרוש על מנת ליצור אנימציה אשר ימומש במחלקת Animation.

בנוסף, יצרתי מילון TextureDictionray אשר תפקידו להעלות את כל הטקסטורות ולשמור אותן במילון זה. דבר זה יאפשר גישה קלה לכל הטקסטורות הקיימות בפרויקט. מנגנון זה עובד בעזרת ה-enum אשר הגדרתי במחלקת General ובו אני מגדיר אילו תיקיות על המילון להיכנס ולשמור את כל הטקסטורות שם.

```
using System.Linq;
using System.Collections.Generic;
using System.IO;
namespace Slugs
{
    static class Dictionaries
    {
        public static Dictionary<FolderTextures, Dictionary<string, Texture2D>>
TextureDictionary;
        public static Dictionary<Heroes, Dictionary<PlayerState, SpriteSheet>>
PlayerAnimationDictionary;

        public static void AnimationDicInit()
        {
            PlayerAnimationDictionary =
```

```
            new Dictionary<Heroes, Dictionary<PlayerState, SpriteSheet>>();

            foreach (Heroes hero in Enum.GetValues(typeof(Heroes)))
            {
                Dictionary<PlayerState, SpriteSheet> heroDic = new
Dictionary<PlayerState, SpriteSheet>();
                foreach (PlayerState state in Enum.GetValues(typeof(PlayerState)))
                {
                    if (File.Exists(Directory.GetCurrentDirectory() + "/Content/" +
hero + "/" + state + ".xnb"))
                    {
                        heroDic.Add(state, new SpriteSheet(hero, state));
                    }
                }
                PlayerAnimationDictionary.Add(hero, heroDic);
            }
        }
        public static void TexturesDicInit()
        {
            TextureDictionary = new Dictionary<FolderTextures, Dictionary<string,
Texture2D>>();
            Console.WriteLine("\n  enviroment path " +
Environment.CurrentDirectory);
            string path = Directory.GetCurrentDirectory() + "/Content/ScreenAddons";
            Console.WriteLine("\n  path " + path);

            foreach (FolderTextures folder in
Enum.GetValues(typeof(FolderTextures)))
            {
                string[] filesInFolder = Directory.GetFiles(path + "/" + folder);
                Dictionary<string, Texture2D> folderDic = new Dictionary<string,
Texture2D>();

                int i = 0;
                while (i < filesInFolder.Length)
                {
                    //Path.
                    filesInFolder[i] =
Path.GetFileNameWithoutExtension(filesInFolder[i]);
                    Console.WriteLine("\n   filesInFolder[" + i + "] : " +
filesInFolder[i]);
                    // textures have a Name field.
                    Texture2D textureInFolder = General.cm.Load<Texture2D>(path +
"/" + folder + "/" + filesInFolder[i]);
                    Console.WriteLine("    t.name : " + textureInFolder.Name + "
t.ToString() : " + textureInFolder.ToString());

                    folderDic.Add(filesInFolder[i], textureInFolder);
                    // next file
                    i++;
                }
                TextureDictionary.Add(folder, folderDic);
            }
        }
    }
}
```

מחלקת Animation

מחלקה זו יורשת את מחלקת Drawable וזאת כיוון שכל אנימציה שנוצרת ניתן לצייר על המסך ולכן קיימות לה כל התכונות הדרושות לציור. מחלקת Animation אחראית ליצירת האנימציה הרצויה בהתאם לדמות ולמצב הנבחר.

פעולות:

- בנאים המאתחלים את האנימציה לפריים הראשון ומעדכנים למצב הנבחר של אותה הדמות.
- פעולת עדכון לפריים הבא או חזרה לראשון בהתאם לתכונות ולפעולות על המחלקה.

```
namespace Slugs
{
    public class Animation : Drawable
    {
        public SpriteSheet SpriteSheetAnimation { get; set; }
        public int CurrentIndex { get; set; }
        public int FrameDelay { get; set; }
        public Animation(Heroes hero, PlayerState state, Vector2 position,
                         Color color, float rotation, float scale, SpriteEffects
effects, float layerDepth) :
                         base(position, color, rotation, scale, effects, layerDepth)
        {
            SpriteSheetAnimation =
Dictionaries.PlayerAnimationDictionary[hero][state];
            CurrentIndex = 0;
            FrameDelay = 0;
            Texture = SpriteSheetAnimation.Tex;
        }
        public Animation(Heroes hero, PlayerState state, Vector2 position, Color
color, float rotation, float scale, float layerDepth) :
                         base(color, rotation, scale, layerDepth)
        {
            SpriteSheetAnimation =
Dictionaries.PlayerAnimationDictionary[hero][state];
            CurrentIndex = 0;
            FrameDelay = 0;
            Texture = SpriteSheetAnimation.Tex;
        }

        public void Update()
        {
            Texture = SpriteSheetAnimation.Tex;
            SourceRectangle = SpriteSheetAnimation.Recs[CurrentIndex];
            if (Effects == SpriteEffects.FlipHorizontally)
                Origin = new Vector2(SourceRectangle.Value.Width -
SpriteSheetAnimation.Orgs[CurrentIndex].X,
SpriteSheetAnimation.Orgs[CurrentIndex].Y);
            else
                Origin = SpriteSheetAnimation.Orgs[CurrentIndex];
            if ((int)SpriteSheetAnimation.Pace < ++FrameDelay)
            {
                FrameDelay = 0;
                CurrentIndex++;
                CurrentIndex %= SpriteSheetAnimation.Recs.Count;
            }
        }
    }}
```

המחלקה האבסטרקטית BaseKeys

מחלקה זו נועדה להגדיר את הפעולות של כפתורים השייכים לפרויקט.

מחלקת UserKeys

מחלקה זו יורשת את מחלקת BaseKeys ואחראית על הכפתורים השייכים לשחקן במשחק.
המחלקה מממשת את הפעולות שהוגדרו מחלקת BaseKeys.

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

namespace Slugs
{
    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 QPressed();
        public abstract Boolean EPressed();
        public abstract Boolean QReleased();
        public abstract Boolean EReleased();

        public abstract Boolean EnterPressed();
        public abstract Boolean SpacePressed();
        public abstract Boolean EnterReleased();
        public abstract Boolean SpaceReleased();

        public abstract Boolean OnePressed();
        public abstract Boolean TwoPressed();
        public abstract Boolean ThreePressed();

        public abstract void Update();
    }

    public class UserKeys : BaseKeys
    {
        #region Data
        Keys right, left, up, down, enter, Q, E;
        #endregion

        #region Ctors
        public UserKeys(Keys right, Keys left, Keys up, Keys down, Keys enter, Keys
q, Keys e)
        {
            this.right = right;
            this.left = left;
            this.up = up;
```

```
        this.down = down;
        this.enter = enter;
        this.Q = q;
        this.E = e;
    }
    #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 bool QPressed()
    {
        return Keyboard.GetState().IsKeyDown(Q);
    }

    public override bool EPressed()
    {
        return Keyboard.GetState().IsKeyDown(E);
    }

    public override bool QReleased()
    {
        return Keyboard.GetState().IsKeyUp(Q);
    }

    public override bool EReleased()
    {
        return Keyboard.GetState().IsKeyUp(E);
    }

    public override bool EnterPressed()
```

```
        {
            return Keyboard.GetState().IsKeyDown(Keys.Enter);
        }

        public override bool SpacePressed()
        {
            return Keyboard.GetState().IsKeyDown(Keys.Space);
        }

        public override bool EnterReleased()
        {
            return Keyboard.GetState().IsKeyUp(Keys.Enter);
        }

        public override bool SpaceReleased()
        {
            return Keyboard.GetState().IsKeyUp(Keys.Space);
        }

        public override void Update()
        {
            return;
        }

        public override bool OnePressed()
        {
            return Keyboard.GetState().IsKeyDown(Keys.D1);
        }

        public override bool TwoPressed()
        {
            return Keyboard.GetState().IsKeyDown(Keys.D2);
        }

        public override bool ThreePressed()
        {
            return Keyboard.GetState().IsKeyDown(Keys.D3);
        }
    }
}
```

המחלקה 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 המסך עם כל התכונות הנחוצות לציור.
- עדכון – עדכון הכפתור בהתאם למצב העכבר – האם לחוץ, האם העכבר מרחף מעל הכפתור וכו'.
- בנאים לסוגי כפתורים שונים.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Linq;
using System.Collections.Generic;
namespace Slugs
{
    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 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 string Text { get; set; }

        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)
        {
            spriteBatch.Draw(texture, Position, null, color, 0, new
Vector2(texture.Width / 2, texture.Height / 2), Scale, SpriteEffects.None, 0);

            if (!string.IsNullOrEmpty(Text))
            {
                var x = (Rectangle.X + (Rectangle.Width / 2)) -
(font.MeasureString(Text).X / 2);
                var y = (Rectangle.Y + (Rectangle.Height / 2)) -
(font.MeasureString(Text).Y / 2);

                spriteBatch.DrawString(font, Text, new Vector2(x, y), PenColor);
            }
        }

        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());
                }
            }
        }
    }
}
```

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.

המחלקה האבסטרקטית State

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.

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.

מחלקת MenuState

מחלקה זו אחראית על מסך הפתיחה של המשחק ומציגה בפני השחקן אפשרויות לבחירת מספר הקבוצות ומספר השחקנים בכל קבוצה. כמו כן, קיימים שני כפתורים – התחלת משחק ויציאה.

פעולות:

- בנאי MenuState - כאן נוצרים כל הכפתורים עבור בחירת מספר השחקנים, מספר הקבוצות וכפתורי ההתחלה והיציאה.
- פעולות להתמודדות עם לחיצות על כפתורי המספרים והתחלת וסיום המשחק.
- עדכון – עדכון הכפתורים בהתאם ללחיצות העכבר ומיקומו.
- ציור – ציור כל הרקע, הטקסטורות והכפתורים במסך הפתיחה.
- אחרי עדכון – אחראית על שחרור משאבים שאינם נחוצים עוד.


```
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 Slugs.ProjectStates
{
    public class MenuState : State
```

```
    {
        private List<Component> components;
        public List<Button> teamButtons;
        public List<Button> playerButtons;
        private int numOfTeams = 2;
        private int numOfPlayers = 2;

        public MenuState(Game1 game) : base(General.cm, General.device, game)
        {

            var buttonTexture =
Dictionaries.TextureDictionary[FolderTextures.Controls]["Button"];
            var buttonFont = General.font;

            var twoButton = new Button(buttonTexture, General.font)
            {
                Scale = 3f,
                Position = new Vector2(650, 300),
                Text = "2",
            };
            twoButton.Click += TwoGameButton_Click;

            var threeButton = new Button(buttonTexture, General.font)
            {
                Scale = 3f,
                Position = new Vector2(700, 300),
                Text = "3",
            };

            threeButton.Click += ThreeGameButton_Click;

            var fourButton = new Button(buttonTexture, General.font)
            {
                Scale = 3f,
                Position = new Vector2(750, 300),
                Text = "4",
            };
            fourButton.Click += FourGameButton_Click;

            var newGameButton = new
Button(Dictionaries.TextureDictionary[FolderTextures.Controls]["StartGameButton"])
            {
                Scale = 1f,
                Position = new Vector2(General.screenWidth / 2, 450)
            };
            newGameButton.Click += NewGameButton_Click;

            var quitGameButton = new
Button(Dictionaries.TextureDictionary[FolderTextures.Controls]["QuitGameButton"])//,
buttonFont)
            {
                Position = new Vector2(General.screenWidth / 2, 550),
                Scale = 1f
            };

            quitGameButton.Click += QuitGameButton_Click;
            components = new List<Component>()
            {
                newGameButton,
                quitGameButton
            };
            teamButtons = new List<Button>()
```

```
            {
                twoButton,
                threeButton,
                fourButton
            };

            Button twoB = new Button(twoButton);
            Vector2 pos = twoButton.Position;
            pos.Y = 350;
            twoB.Position = pos;

            Button threeB = new Button(threeButton);
            pos = threeB.Position;
            pos.Y = 350;
            threeB.Position = pos;

            Button fourB = new Button(fourButton);
            pos = fourB.Position;
            pos.Y = 350;
            fourB.Position = pos;

            twoB.Click += TwoPGameButton_Click;
            threeB.Click += ThreePGameButton_Click;
            fourB.Click += FourPGameButton_Click;

            playerButtons = new List<Button>()
            {
                twoB,
                threeB,
                fourB
            };

            twoButton.Clicked = true;
            twoButton.color = Color.Gray;
            twoB.Clicked = true;
            twoB.color = Color.Gray;
        }

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

spriteBatch.Draw(Dictionaries.TextureDictionary[FolderTextures.Graphics]["SlugsWallp
aper"], new Vector2(0, 0), Color.White);

spriteBatch.Draw(Dictionaries.TextureDictionary[FolderTextures.Graphics]["SlugsLogo"
], new Vector2(350, 50), Color.White);

spriteBatch.Draw(Dictionaries.TextureDictionary[FolderTextures.Controls]["NumOfTeams
"], new Vector2(260, 270), Color.White);

spriteBatch.Draw(Dictionaries.TextureDictionary[FolderTextures.Controls]["NumOfPlaye
rs"], new Vector2(260, 330), Color.White);

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

            foreach (var button in playerButtons)
                button.Draw(gameTime, spriteBatch);
        }
```

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

public void TwoGameButton_Click(object sender, EventArgs e)
{
    foreach (var button in teamButtons)
        if (button.Text == "2")
        {
            button.color = Color.Gray;
            button.Clicked = true;
            numOfTeams = 2;
        }
        else
        {
            button.color = Color.White;
            button.Clicked = false;
        }
}

public void ThreeGameButton_Click(object sender, EventArgs e)
{
    foreach (var button in teamButtons)
        if (button.Text == "3")
        {
            button.color = Color.Gray;
            button.Clicked = true;
            numOfTeams = 3;
        }
        else
        {
            button.color = Color.White;
            button.Clicked = false;
        }
}

public void FourGameButton_Click(object sender, EventArgs e)
{
    foreach (var button in teamButtons)
        if (button.Text == "4")
        {
            button.color = Color.Gray;
            button.Clicked = true;
            numOfTeams = 4;
        }
        else
        {
            button.color = Color.White;
            button.Clicked = false;
        };
}

public void TwoPGameButton_Click(object sender, EventArgs e)
{
    foreach (var button in playerButtons)
        if (button.Text == "2")
        {
            button.color = Color.Gray;
            button.Clicked = true;
            numOfPlayers = 2;
        }
```

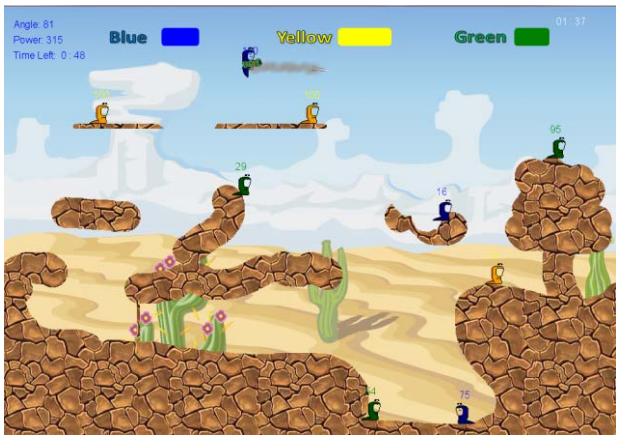
```
                else
                {
                    button.color = Color.White;
                    button.Clicked = false;
                }
            }

            public void ThreePGameButton_Click(object sender, EventArgs e)
            {
                foreach (var button in playerButtons)
                    if (button.Text == "3")
                    {
                        button.color = Color.Gray;
                        button.Clicked = true;
                        numOfPlayers = 3;
                    }
                    else
                    {
                        button.color = Color.White;
                        button.Clicked = false;
                    }
            }
            public void FourPGameButton_Click(object sender, EventArgs e)
            {
                foreach (var button in playerButtons)
                    if (button.Text == "4")
                    {
                        button.color = Color.Gray;
                        button.Clicked = true;
                        numOfPlayers = 4;
                    }
                    else
                    {
                        button.color = Color.White;
                        button.Clicked = false;
                    };
            }
            public override void PostUpdate(GameTime gameTime)
            {
            }
            public override void Update(GameTime gameTime)
            {
                foreach (var component in components)
                    component.Update(gameTime);

                foreach (var button in teamButtons)
                    button.Update(gameTime);

                foreach (var button in playerButtons)
                    button.Update(gameTime);
            }
            public void QuitGameButton_Click(object sender, EventArgs e)
            {
                Console.WriteLine("Exit");
                game.Exit();
            }
        }
    }
```

מחלקת GameState

מחלקה זו אחראית למעשה למהלך המשחק. כל הפעולות וכל הדברים המאפשרים למשחק להתחולל קורים כאן.

פעולות:

- בנאי GameState – בנאי זה מזומן עם לחיצת הכפתור להתחלת המשחק ומאתחל את כל המשתנים של המשחק, אדמת המשחק, יצירת השחקנים והשמתם במפה בהתאם לבחירת הקבוצות ומספר השחקנים במסך הפתיחה. כמו כן, המצלמה מאותחלת לעקוב אחר השחקן המתחיל.
- SetUpPlayers – פעולה זו יוצרת את הקבוצות והשחקנים בתוכן בהתאם לבחירת השחקן.
- AreAllPlayersOnGround – בדיקה אם כל השחקנים נמצאים על האדמה.
- HandleShootableCollisions – יצירת פיצוצים ועדכון אדמת המשחק ביריות שקיימים התנגשויות.
- CheckShootableCollisions - בדיקה ועדכון על ההתנגשויות של היריות.
- עדכון – עדכון כל השחקנים, המפה וכו'.
- ציור – ציור כל הרקע, הטקסטורות, מפת המשחק, השחקנים שעליה והמידע הטקסטואלי של המשחק.
- DrawPlayers –ציור כל השחקנים והמידע שלהם.
- DrawText – ציור כל הטקסט שקיים על המסך – זמן נותר, זוויות נשק, עוצמה.
- DrawTeamsTotalHealth – ציור החיים הכוללים של כל קבוצה.
- HandleWin – בדיקה אם המשחק הסתיים – ניצחון או תיקו.
- CheckIsWin – בדיקה אם הקבוצה שנשלחה ניצחה או לא.
- ActivateWinMode / ActivateTeMode –פעולות המזומנות בהתאם למצב סוף המשחק.
- אחרי עדכון – אחראית על שחרור משאבים שאינם נחוצים עוד.




```
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 Slugs.ProjectStates
{
    public class GameState : State
    {
        Color[] teamColors;

        public Drawable background;
        public int numOfPlayersInTeam;
        public int numOfTeams;
        public bool isNext;
        int TotalTeamHealth;


        public GameState(Game1 game, int numOfPlayersInTeam, int numOfTeams) :
base(General.cm, General.device, game)
        {
```

```
            background = new
Drawable(Dictionaries.TextureDictionary[FolderTextures.Graphics]["Desert"], new
Vector2(0, 0), null, Color.White, 0, Vector2.Zero, 1, SpriteEffects.None, 0);

            isNext = false;
            this.numOfPlayersInTeam = numOfPlayersInTeam;
            this.numOfTeams = numOfTeams;
            TotalTeamHealth = 100 * numOfPlayersInTeam;
            SetUpPlayers();
            General.cam = new Camera(General.players[0]);

            General.GenerateTerrainMask();


Foreground.Create(Dictionaries.TextureDictionary[FolderTextures.Graphics]["terrain3"
], Dictionaries.TextureDictionary[FolderTextures.Graphics]["groundSlug"], new
Rectangle(0, 0, General.screenWidth, General.screenHeight), Color.White);


Explosion.Create(Dictionaries.TextureDictionary[FolderTextures.Graphics]["explosion"
]);
        }

        public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
        {
            spriteBatch.End();
            spriteBatch.Begin(SpriteSortMode.Deferred, null, null, null, null, null,
General.cam.Mat);

            background.Draw();
            Foreground.Draw();

            DrawPlayers();

            spriteBatch.End();
            spriteBatch.Begin();
            General.players[General.currentPlayer].DrawPlayTime();

            DrawText(gameTime);
            DrawTeamsTotalHealth();

            spriteBatch.End();

            spriteBatch.Begin(SpriteSortMode.Deferred, BlendState.Additive, null,
null, null, null, General.cam.Mat);
            Explosion.Draw();

            spriteBatch.End();
        }

        public void SetUpPlayers()
        {
            General.currentPlayer = 0;
            int numOfPlayers = numOfPlayersInTeam * numOfTeams;
            General.numberOfPlayers = numOfPlayers;
            Color teamColor;
            teamColors = new Color[4];
            teamColors[0] = Color.Blue;
            teamColors[1] = Color.Yellow;
            teamColors[2] = Color.Green;
            teamColors[3] = Color.Red;
```

```
            int team = 0;
            General.players = new Slug[numOfPlayers];
            for (int i = 0; i < numOfTeams; i++)
            {

                for (int j = 0; j < numOfPlayersInTeam; j++)
                {
                    teamColor = teamColors[team];
                    Vector2 playerPosition = new Vector2();
                    playerPosition.X = General.screenWidth / (numOfPlayers + 1) * (i
* numOfPlayersInTeam + j + 1);
                    playerPosition.Y = 100;//
General.terrainContour[(int)playerPosition.X];
                    if (i == 0 && j == 0)
                        General.players[i * numOfPlayersInTeam + j] = new
Slug(Heroes.Slug, playerPosition, true, teamColor, team,
Dictionaries.TextureDictionary[FolderTextures.Graphics]["rocket"], 0, 1f, 0, Keys.A,
Keys.D, Keys.W, Keys.S, Keys.Enter, Keys.Q, Keys.E, false);
                    else
                        General.players[i * numOfPlayersInTeam + j] = new
Slug(Heroes.Slug, playerPosition, false, teamColor, team,
Dictionaries.TextureDictionary[FolderTextures.Graphics]["rocket"], 0, 1f, 0, Keys.A,
Keys.D, Keys.W, Keys.S, Keys.Enter, Keys.Q, Keys.E, false);

                    team++;
                    team = team % numOfTeams;
                }

            }

            General.currentTeamHealth = new int[numOfTeams];
            for (int i = 0; i < numOfTeams; i++)
            {
                General.currentTeamHealth[i] = TotalTeamHealth;
            }
        }

        public override void PostUpdate(GameTime gameTime)
        {
        }

        public override void Update(GameTime gameTime)
        {
            foreach (Slug player in General.players)
                if (player.IsAlive)
                    player.UpdateState();

            if (isNext)
            {
                if (AreAllPlayersOnGround())
                {
                    General.NextPlayer();
                    isNext = false;
                }
            }

            if (General.players[General.currentPlayer].IsShot)
                HandleShootablesCollision(gameTime);

            if (Explosion.particleList.Count > 0)
                Explosion.UpdateParticles(gameTime);
```

```
            HandleWin();

            General.cam.Update();
        }

public bool AreAllPlayersOnGround()
        {
            bool isOnGround = true;
            foreach (Slug player in General.players)
                if (player.IsAlive)
                    if (!player.IsOnGround())
                        isOnGround = false;
            return isOnGround;
        }


        public void HandleShootablesCollision(GameTime gameTime)
        {
            isNext = true;
            List<Shootable> shootablesToRemove = new List<Shootable>();

            foreach (Shootable shootable in
General.players[General.currentPlayer].PWeapon.ListOfShootables)
            {
                if (!shootable.IsOutOfScreen())
                {
                    Vector2 collisionPos = CheckShootableCollisions(gameTime,
shootable);

                    if (collisionPos.X > -1)
                    {

General.players[General.currentPlayer].RemoveWeaponShootableAndUpdate(shootable);

                        shootable.AddExplosion(collisionPos, gameTime);

                        foreach (Slug player in General.players)
                        {
                            Vector2 explosionVector = (player.Position + new
Vector2(0,-10)) - collisionPos;
                            if (explosionVector.Length() <= shootable.Damage)
                            {
                                player.UpdateHealth((int)(shootable.Damage -
explosionVector.Length()));
                                player.velocityX = shootable.powerOfExplosion *
explosionVector.X / Math.Abs(explosionVector.X);//5 * explosionVector.X /
Math.Abs(explosionVector.X);
                                player.velocityY = shootable.powerOfExplosion *
explosionVector.Y / Math.Abs(explosionVector.Y);//5 * explosionVector.Y /
Math.Abs(explosionVector.Y);
                            }
                        }
                        break;
                    }
                }
                else
                {

General.players[General.currentPlayer].RemoveWeaponShootableAndUpdate(shootable);
                    break;
                }
            }
```

```
            if
(General.players[General.currentPlayer].PWeapon.ListOfShootables.Count == 0)
                isNext = true;
            else
                isNext = false;
        }

        public Vector2 CheckShootableCollisions(GameTime gameTime, Shootable
shootable) // TODO: if there's no time but a rocket is flying in the air it should
let it finish.
        {

            Vector2 collisionPos = new Vector2(-1, -1);
            Vector2 terrainCollisionPoint =
Foreground.CheckTerrainCollision(shootable);

            // If the shootable is in the map

                // If the shootable did not collide with the terrain check the
collision with each player.
                if (terrainCollisionPoint.X == -1)
                {
                    foreach (Slug player in General.players)
                    {
                        if (player != General.players[General.currentPlayer])
                        {
                            Vector2 playerCollisionPoint =
player.CheckPlayerCollision(shootable);
                            if (playerCollisionPoint.X > -1)
                            {
                                collisionPos = playerCollisionPoint;
                                break;
                            }
                        }
                    }
                }
                else
                {
                    collisionPos = terrainCollisionPoint;
                }

            return collisionPos;
        }

public void DrawPlayers()
        {
            foreach (Slug player in General.players)
                if (player.IsAlive)
                    player.Draw();
        }

        public void DrawText(GameTime gameTime)
        {
            int currentAngle =
(int)MathHelper.ToDegrees(General.players[General.currentPlayer].WeaponAngle);
            General.sb.DrawString(General.font, "Angle: " + currentAngle.ToString(),
new Vector2(20, 20), General.players[General.currentPlayer].color);
            General.sb.DrawString(General.font, "Power: " +
General.players[General.currentPlayer].Power.ToString(), new Vector2(20, 45),
General.players[General.currentPlayer].color);
```

```
            int minute = gameTime.TotalGameTime.Minutes, second =
gameTime.TotalGameTime.Seconds;
            string min = minute.ToString(), sec = second.ToString();
            if (minute < 10 && minute >= 0)
                min = '0' + min;

            if (second < 10 && second >= 0)
                sec = '0' + sec;

            General.sb.DrawString(General.font, min + " : " + sec, new
Vector2(General.screenWidth - 100, 15), Color.White);
        }

        public void DrawTeamsTotalHealth()
        {
            Texture2D X =
Dictionaries.TextureDictionary[FolderTextures.Graphics]["X"];
            Texture2D healthLaneTexture =
Dictionaries.TextureDictionary[FolderTextures.Graphics]["HealthLane"];
            for (int i = 0; i < numOfTeams; i++)
            {
                int place = (General.screenWidth - 140) / numOfTeams;
                int posX = 120 + place * i;

                Texture2D teamColorNameTexture =
Dictionaries.TextureDictionary[FolderTextures.Graphics][General.teamColorString[i]];


                General.sb.Draw(teamColorNameTexture, new Rectangle(posX, 20,
teamColorNameTexture.Width, teamColorNameTexture.Height - 10), Color.White);
                if (General.currentTeamHealth[i] > 0)
                {
                    double percentage = (double)General.currentTeamHealth[i] /
(double)TotalTeamHealth;
                    General.sb.Draw(healthLaneTexture, new Rectangle(posX + 140, 34,
(int)((healthLaneTexture.Width * percentage)), healthLaneTexture.Height - 10),
teamColors[i]);
                }
                else
                {
                    General.sb.Draw(X, new Rectangle(posX + 140, 30, X.Width - 20,
X.Height - 20), Color.White);
                }
            }
        }

        private void ActivateWinMode(int numOfTeam)
        {
            game.ChangeState(new EndState(game, this, numOfTeam + 1));
        }

        private void ActivateTieMode()
        {
            game.ChangeState(new EndState(game, this, 0));
        }

        private void HandleWin()
        {
            int countZeroHealth = 0;
            for (int i = 0; i < numOfTeams; i++)
            {
                if (General.currentTeamHealth[i] > 0 && CheckIsWin(i))
```

```
                ActivateWinMode(i);
            else if (General.currentTeamHealth[i] == 0)
                countZeroHealth++;
            else
                break;
        }

        if (countZeroHealth == numOfTeams)
            ActivateTieMode();
    }


    private bool CheckIsWin(int numOfTeam)
    {
        for (int i = 0; i < numOfTeams; i++)
        {
            if (General.currentTeamHealth[i] > 0 && i != numOfTeam)
                return false;
        }
        return true;
    }
}
}
```

מחלקת EndState

מחלקה זו אחראית על מסך הסיום, ומציגה את מצב המשחק בסופו – ניצחון של אחת הקבוצות או תיקו (כל הקבוצות מתו). בנוסף, על מנת ליצור מצב בו המשחק ממשיך לפעול ולא רק להציג מסך סיום, השתמשתי במצב המשחק והמשכתי לעדכן ולצייר חלקים מן המשחק ועל שכבה זו ציירתי ריבוע שחור עם בהירות אשר יצר מצב בו נראה כאילו למסך ירד הבהירות ומאפשר לצייר על שכבה זו את מצב סיום המשחק ובנוסף כפתורי סיום או התחלת משחק חדש.

פעולות:

- בנאי EndState - כאן נוצרים שני הכפתורים יציאה והתחלת משחק חדש.
- פעולות להתמודדות עם לחיצות על כפתורי התחלת וסיום המשחק.
- עדכון – עדכון הכפתורים בהתאם ללחיצות העכבר ומיקומו.
- ציור – ציור כל הרקע, הטקסטורות והכפתורים במסך הסיום, כמו כן, כאן נבדק מהו מצב סיום המשחק – ניצחון של אחת הקבוצות או תיקו ומצויר בהתאם.
- אחרי עדכון – אחראית על שחרור משאבים שאינם נחוצים עוד.


```
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 Slugs.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(General.cm, General.device, game)
        {
            if (numOfTeamWon == 0)
                isTie = true;
            else
                this.numOfTeamWon = numOfTeamWon - 1;

            this.gameState = gameState;

            var buttonTexture =
Dictionaries.TextureDictionary[FolderTextures.Controls]["Button"];
            var buttonFont = General.font;

            var newGameButton = new
Button(Dictionaries.TextureDictionary[FolderTextures.Controls]["NewGameButton"])
            {
                Scale = 1f,
                Position = new Vector2(General.screenWidth / 2, 500)
            };

            newGameButton.Click += NewGameButton_Click;
```

```
            var quitGameButton = new
Button(Dictionaries.TextureDictionary[FolderTextures.Controls]["QuitGameButton"])//,
buttonFont)
            {
                Position = new Vector2(General.screenWidth - 100, 600),
                Scale = 1f
            };

            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,
General.cam.Mat);

            gameState.background.Draw();
            Foreground.Draw();

            gameState.DrawText(gameTime);

            gameState.DrawPlayers();
            gameState.DrawTeamsTotalHealth();

            spriteBatch.End();

            spriteBatch.Begin(SpriteSortMode.Deferred, BlendState.Additive, null,
null, null, null, General.cam.Mat);
            Explosion.Draw();

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


spriteBatch.Draw(Dictionaries.TextureDictionary[FolderTextures.Graphics]["DarkScreen
"], new Vector2(0, 0), Color.White * 0.5f);

            if (isTie)
            {
                Texture2D tieTexture =
Dictionaries.TextureDictionary[FolderTextures.Graphics]["Tie"];
                spriteBatch.Draw(tieTexture, new Vector2(General.screenWidth / 2 -
tieTexture.Width / 2, 50), Color.White);
            }
            else
            {
                Texture2D wonTexture =
Dictionaries.TextureDictionary[FolderTextures.Graphics][General.teamColorString[numO
fTeamWon] + "Won"];
                spriteBatch.Draw(wonTexture, new Vector2(General.screenWidth / 2 -
wonTexture.Width / 2, 50), Color.White);

            }
```

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

        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 (Slug player in General.players)
                if (player.IsAlive)
                    player.UpdateState();

            if (Explosion.particleList.Count > 0)
                Explosion.UpdateParticles(gameTime);

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

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

מערכת הנשקים במשחק:

חלק שהיה חשוב לי לבצע אותו בצורה מודולרית ככל האפשר אשר תאפשר שימוש ויישום קל של מחלקות שונות והאינטראקציה בינהן הינו מערכת הנשקים.

על מנת ליצור נשקים שונים בצורה יעיעלה ונוחה ניסיתי לפשט כמה שניתן את הגדרתו של כלי נשק:

כל נשק הינו למעשה כלי אשר מכיל בתוכו סוג ירייה מסויימת בכמות מסויימת המשתנה בין הנשקים. וכך כל ירייה בעלת התנהגות שונה והשפעה שונה על אובייקטים. כך למשל, טיל הנורה על ידיי בזוקה הינו בעל טווח פיצוץ גדול ומושפע רבות מגרביטציית המשחק (כך קבעתי במשחק שלי). לעומת זאת, תת-מקלע יורה צרור כדורים אשר להם השפעה קטנה על אובייקטים ופיצוץ קטן עם נזק מינימלי וגם כוח המשיכה זניח עבורם (נעים בקו ישר).

עם פישוט כלי הנשק והתנהגותם הסקתי מסקנות אשר אפשרו לי ליצור מחלקות שיפעלו באופן המתאים לכל נשק והתנהגותו המיוחדת.

להלן תיאור המחלקות:

המחלקה האבסטרקטית Shootable

זוהי המחלקה המייצגת כל ירייה (יורשת Drawable) אשר נורית מנשק כלשהו כמו קליע או טיל. במחלקה זו יצרתי כמה פעולות אשר ממומשות ישירות כאן כיוון שהתאימו להתנהגות כל סוג ירייה, כמו למשל, בדיקה אם מחוץ למסך או מציאת הראש שלה.

פעולות:

- בנאי Shootable – מאתחל ערכים ויוצר ירייה.
- GetShootableNose – מציאת ראש הירייה.
- AddExplosion – פעולה אבסטרקטית אשר יוצרת פיצוץ.
- IsOutOfScreen – פעולה זו בודקת אם הירייה מחוץ למסך.
- Update – פעולה אבסטרקטית אשר מעדכנת את הירייה.
- Reset – פעולה אבסטרקטית אשר מאפסת את הירייה.

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

namespace Slugs
{
    public abstract class Shootable : Drawable
    {
        public float Damage { get; set; }
        public Vector2 Direction { get; set; }
        public bool IsFlying { get; set; }
        public float Angle { get; set; }
        public int TimeToActivate { get; set; }
        public int powerOfExplosion { get; set; }
```

```
        public Shootable(Vector2 direction, float damage, Texture2D texture, Vector2
position, Rectangle? sourceRectangle,
                         Color color, float rotation, Vector2 origin, float scale,
SpriteEffects effects, float layerDepth)
            : base(texture, position, sourceRectangle,
                        color, rotation, origin, scale, effects, layerDepth)
        {
            Damage = damage;
            Direction = direction;
        }

        public Vector2 GetShootableNose()
        {
            Vector2 newOrigin = Vector2.Transform(new Vector2(0, -this.Origin.Y),
Matrix.CreateRotationZ(this.Angle));
            newOrigin *= this.Scale;

            Vector2 pos = this.Position;
            pos += newOrigin;

            return pos;
        }

        public abstract void AddExplosion(Vector2 collisionPos, GameTime gameTime);

        public bool IsOutOfScreen()
        {
            bool isOutOfScreen = Position.Y > General.screenHeight;
            isOutOfScreen |= Position.X < 0;
            isOutOfScreen |= Position.X > General.screenWidth;

            return isOutOfScreen;
        }

        public abstract void Update(Vector2 position, float angle);

        public abstract void Reset();
    }
}
```

מחלקת Bullet

מחלקה זו יורשת Shootable ואחראית על כדורים הנורים מרובים כגון תת-מקלע ושוטגאן.
כדורים מסוג זה בעלי נזק די נמוך וכמו כן גם פיצוץ ברדיו ס קטן.

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

namespace Slugs
{
    public class Bullet : Shootable
    {
        public Bullet(Texture2D texture, Vector2 position, Rectangle?
sourceRectangle,
                        Color color, float rotation, Vector2 origin, float scale,
SpriteEffects effects, float layerDepth, Vector2 direction, float damage)
            : base(direction, damage, texture, position, sourceRectangle,
                        color, rotation, origin, scale, effects, layerDepth)
        {
            Damage = damage;
            powerOfExplosion = 1;
        }
        public override void Update(Vector2 position, float angle)
        {
            if (TimeToActivate <= General.Time)
            {
                IsFlying = true;
                Position += Direction;
                Angle = (float)Math.Atan2(Direction.X, -Direction.Y);
            }
            else
            {
                Position = position;
                Angle = angle;
                Vector2 up = new Vector2(0, -1);
                Matrix rotMatrix = Matrix.CreateRotationZ(Angle);
                Direction = Vector2.Transform(up, rotMatrix);
                Direction *= 8f;
            }
        }
        public override void Draw()
        {
            if (IsFlying)
                General.sb.Draw(
                        Texture, Position, SourceRectangle,
                        color, Angle, Origin,
                        Scale, Effects, 0);
        }
        public override void Reset()
        {
        }
        public override void AddExplosion(Vector2 collisionPos, GameTime gameTime)
        {
            Explosion.AddExplosion(collisionPos, 1, 20.0f, 10.0f, gameTime);
        }
    }}
```

מחלקת Rocket

מחלקה זו יורשת Shootable ואלראית על טיל הנורה מבזוקה. לטיל נזק ורדיוס פיצוץ די גדולים.

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

namespace Slugs
{
    public class Rocket : Shootable
    {
        public List<Vector2> SmokeList { get; set; }

        public Rocket(Texture2D texture, Vector2 position, Rectangle?
sourceRectangle,
                      Color color, float rotation, Vector2 origin, float scale,
SpriteEffects effects, float layerDepth, Vector2 direction, float damage)
            : base(direction, damage, texture, position, sourceRectangle,
                      color, rotation, origin, scale, effects, layerDepth)
        {
            SmokeList = new List<Vector2>();
            Damage = damage;
            powerOfExplosion = 5;
        }

        public override void Update(Vector2 position, float angle)
        {
            if (TimeToActivate <= General.Time)
            {
                IsFlying = true;
                Vector2 gravity = new Vector2(0, 1);
                Direction += gravity / 10.0f;
                Position += Direction;
                Angle = (float)Math.Atan2(Direction.X, -Direction.Y);

                for (int i = 0; i < 5; i++)
                {
                    Vector2 smokePos = Position;
                    smokePos.X += General.randomizer.Next(10) - 5;
                    smokePos.Y += General.randomizer.Next(10) - 5;
                    SmokeList.Add(smokePos);
                }
            }
            else
            {
                Position = position;
                Angle = angle;
                Vector2 up = new Vector2(0, -1);
                Matrix rotMatrix = Matrix.CreateRotationZ(Angle);
                Direction = Vector2.Transform(up, rotMatrix);
                Vector2 gravity = new Vector2(0, 1);
                Direction += gravity / 10.0f;
            }
        }

        public override void Draw()
        {
```

```
            if (IsFlying)
            {
                General.sb.Draw(
                        Texture, Position, SourceRectangle,
                        color, Angle, Origin,
                        Scale, Effects, 0);
                DrawSmoke();
            }
        }

        private void DrawSmoke()
        {
            foreach (Vector2 smokePos in SmokeList)

General.sb.Draw(Dictionaries.TextureDictionary[FolderTextures.Graphics]["smoke"],
smokePos, null, Color.White, 0, new Vector2(40, 35), 0.2f, SpriteEffects.None, 0);
        }

        public override void Reset()
        {
            SmokeList = new List<Vector2>();
        }

        public override void AddExplosion(Vector2 collisionPos, GameTime gameTime)
        {
            Explosion.AddExplosion(collisionPos, 10, 80.0f, 2000.0f, gameTime);
        }
    }
}
```

המחלקה האבסטרקטית Weapon

זוהי המחלקה אשר אחראית על כל נשק הקיים במשחק (יורשת Drawable). בדומה למחלקת Shootable גם כאן יצרתי כמה פעולות אשר ממומשות במחלקה זו, וזאת כיוון שהן מתאימות לכל נשק קיים.

פעולות:

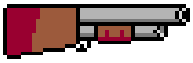
- בנאי Weapon – אשר יוצר מופע של נשק ומאתחל ערכים.
- RemoveShootable – פעולה זו מקבלת ירייה ואחראית להוציא אותה מתוך רשימת היריות – פעולה זו מתבצעת כאשר ירייה יצאה מחוץ למסך או פגעה באובייקט והתפוצצה.
- Reset – פעולה זו אחראית על איפוס הנשק, כלומר מחיקת כל היריות.
- ShootWeapon – פעולה אבסטרקטית אשר אחראית על ירייה מן הנשק.
- Update – פעולה אבסטרקטית אשר מעדכנת את הנשק.
- Draw – פעולה אשר מציירת את הנשק ואת כל רשימת היריות שנורו כבר מן הנשק.

```
namespace Slugs
{
    public abstract class Weapon : Drawable
    {
        public float Angle { get; set; }
        public bool IsShot { get; set; }
        public List<Shootable> ListOfShootables;
        protected Vector2 weaponShifting;
        public Weapon(Texture2D texture, Vector2 position, Rectangle?
sourceRectangle,
                    Color color, float rotation, Vector2 origin, float scale,
SpriteEffects effects, float layerDepth)
        : base(texture, position, sourceRectangle,
                    color, rotation, origin, scale, effects, layerDepth)
        {}
        public abstract void CheckShootablesCollision();
        public void RemoveShootable(Shootable shootable)
        {
            shootable.Reset();
            ListOfShootables.Remove(shootable);

            if (ListOfShootables.Count == 0)
                IsShot = false;
        }
        public void Reset()
        {
            IsShot = false;
            ListOfShootables = new List<Shootable>();
        }
        public abstract void ShootWeapon(Vector2 slugPos, float weaponAngle, float
power);
        public abstract void Update(Vector2 slugPos, float weaponAngle,
SpriteEffects slugEffects);
        public override void Draw()
        {
            if (IsShot)
                foreach (Shootable shootable in ListOfShootables)
                        shootable.Draw();
            General.sb.Draw(Texture, Position, null, Color.White, Angle, Origin,
0.6f, this.Effects, 0);
        }
    }}
```

מחלקת Bazooka

מחלקה זו יורשת את Weapon ואחראית על תפעול הבזוקה.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace Slugs.Weapons
{
    class Bazooka : Weapon
    {
        Rocket PRocket;

        public Bazooka(Texture2D texture, Vector2 position, float angle, Rectangle?
sourceRectangle, Color color, float rotation, Vector2 origin, float scale,
SpriteEffects effects, float layerDepth) : base(texture, position, sourceRectangle,
color, rotation, origin, scale, effects, layerDepth)
        {
            ListOfShootables = new List<Shootable>();
            PRocket =  new
Rocket(Dictionaries.TextureDictionary[FolderTextures.Graphics]["rocket"],
this.Position, null, Color.White, angle, new Vector2(42, 240), 0.1f,
SpriteEffects.None, 1, new Vector2(0, 0), 80);

        }

        public override void ShootWeapon(Vector2 slugPos, float weaponAngle, float
power)
        {
            IsShot = true;
            Rocket rocket = new Rocket(PRocket.Texture, slugPos, null, Color.White,
PRocket.Angle, PRocket.Origin, PRocket.Scale, PRocket.Effects, PRocket.LayerDepth,
PRocket.Direction, PRocket.Damage);

            rocket.Position = slugPos;
            rocket.Angle = weaponAngle;
            Vector2 up = new Vector2(0, -1);
            Matrix rotMatrix = Matrix.CreateRotationZ(rocket.Angle);
            rocket.Direction = Vector2.Transform(up, rotMatrix);
            rocket.Direction *= power / 50.0f;

            rocket.TimeToActivate = General.Time;
            ListOfShootables.Add(rocket);
        }
        public override void Update(Vector2 position, float weaponAngle,
SpriteEffects slugEffects)
        {
            this.Effects = slugEffects;
            Position = position;
            Angle = weaponAngle;
                foreach (Shootable shootable in ListOfShootables)
                    shootable.Update(position, weaponAngle);
        }
    }
}
```



מחלקת Shotgun

מחלקה זו יורשת את Weapon ואחראית על תפעול השוטגאן.

```
using System.Threading.Tasks;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
namespace Slugs.Weapons
{
    class Shotgun : Weapon
    {
        Bullet PBullet;
        public Shotgun(Texture2D texture, Vector2 position, float angle, Rectangle?
sourceRectangle, Color color, float rotation, Vector2 origin, float scale,
SpriteEffects effects, float layerDepth) : base(texture, position, sourceRectangle,
color, rotation, origin, scale, effects, layerDepth)
        {
            ListOfShootables = new List<Shootable>();
            Position = new Vector2(position.X + 3, position.Y - 10);
            PBullet = new
Bullet(Dictionaries.TextureDictionary[FolderTextures.Graphics]["ShotgunBullet"],
this.Position, null, Color.White, angle, Vector2.Zero, 1f, SpriteEffects.None, 1,
new Vector2(0, 0), 20);
        }
        public override void ShootWeapon(Vector2 slugPos, float weaponAngle, float
power)
        {
            IsShot = true;
            int timeToActivate = General.Time;
            float angle = weaponAngle - 0.04f;
            for (int i = 0; i < 3; i++)
            {
                Bullet bullet = new Bullet(PBullet.Texture, slugPos, null,
Color.White, PBullet.Angle, PBullet.Origin, PBullet.Scale, PBullet.Effects,
PBullet.LayerDepth, PBullet.Direction, PBullet.Damage);
                bullet.Position = slugPos;
                bullet.Angle = angle;
                Vector2 up = new Vector2(0, -1);
                Matrix rotMatrix = Matrix.CreateRotationZ(bullet.Angle);
                bullet.Direction = Vector2.Transform(up, rotMatrix);
                bullet.Direction *= 6f;

                bullet.TimeToActivate = timeToActivate;

                ListOfShootables.Add(bullet);
                angle += 0.04f;
            }

        }
        public override void Update(Vector2 position, float weaponAngle,
SpriteEffects slugEffects)
        {
            this.Effects = slugEffects;
            Position = position;
            Angle = weaponAngle;
            foreach (Shootable shootable in ListOfShootables)
                shootable.Update(position, weaponAngle);
        }
    }
}
```

מחלקת Uzi

מחלקה זו יורשת את Weapon ואחראית על תפעולתת המקלע.

```
using System.Threading.Tasks;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
namespace Slugs.Weapons
{
    class Uzi : Weapon
    {
        Bullet PBullet;

        public Uzi(Texture2D texture, Vector2 position, float angle, Rectangle?
sourceRectangle, Color color, float rotation, Vector2 origin, float scale,
SpriteEffects effects, float layerDepth) : base(texture, position, sourceRectangle,
color, rotation, origin, scale, effects, layerDepth)
        {
            ListOfShootables = new List<Shootable>();
            Position = new Vector2(position.X + 3, position.Y - 10);
            PBullet = new
Bullet(Dictionaries.TextureDictionary[FolderTextures.Graphics]["UziBullet"],
this.Position, null, Color.White, angle, Vector2.Zero, 1f, SpriteEffects.None, 1,
new Vector2(0, 0), 20);
        }

        public override void ShootWeapon(Vector2 slugPos, float weaponAngle, float
power)
        {
            IsShot = true;
            int timeToActivate = General.Time;
            for (int i = 0; i < 5; i++)
            {
                Bullet bullet = new Bullet(PBullet.Texture, slugPos, null,
Color.White, PBullet.Angle, PBullet.Origin, PBullet.Scale, PBullet.Effects,
PBullet.LayerDepth, PBullet.Direction, PBullet.Damage);
                bullet.Position = slugPos;
                bullet.Angle = weaponAngle;
                Vector2 up = new Vector2(0, -1);
                Matrix rotMatrix = Matrix.CreateRotationZ(bullet.Angle);
                bullet.Direction = Vector2.Transform(up, rotMatrix);
                bullet.Direction *= 8f;
                bullet.TimeToActivate = timeToActivate;
                ListOfShootables.Add(bullet);
                timeToActivate += 500;
            }

        }

        public override void Update(Vector2 position, float weaponAngle,
SpriteEffects slugEffects)
        {
            this.Effects = slugEffects;
            Position = position;
            Angle = weaponAngle;
            foreach (Shootable shootable in ListOfShootables)
                shootable.Update(position, weaponAngle);
        }
    }
}
```

מחלקת Arrow

מחלקה האחראית על החץ הנע מעל השחקן.

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

namespace Slugs
{
    class Arrow : Drawable
    {
        Vector2 direction;
        Vector2 beginPos;
        Vector2 endPos;

        public Arrow(Texture2D texture, Vector2 position, Color color, Vector2
origin) : base(texture, position, color, origin)
        {
            position.Y = position.Y - 50f;
            Position = position;
            beginPos = position;
            endPos = position + new Vector2(0, 20);
            direction = new Vector2(0, 0.7f);
        }

        public void Update()
        {
            if (Position.Y >= endPos.Y)
                direction = new Vector2(0, -0.7f);

            if (Position.Y <= beginPos.Y)
                direction = new Vector2(0, 0.7f);

            Position += direction;
        }

        public void PostUpdate(Vector2 pos)
        {
            pos.Y = pos.Y - 50f;
            Position = pos;

            beginPos = pos;
            endPos = pos + new Vector2(0, 20);
        }

        public override void Draw()
        {
            General.sb.Draw(Texture, Position, null, Color.White, 0, Origin, 0.5f,
SpriteEffects.None, 0);
        }
    }
}
```

מחלקת Circle

מחלקה זו אחראית על כל המידע השייך לעיגול ואני נעזר בה ביצירת האנימציה ובדיקת ההתנגשויות בשל אובייקטים בשחקן.

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

namespace Slugs
{
    public class Circle
    {
        public Vector2 Center;
        public float Radius;
        public Circle(Vector2 Center, float Radius)
        {
            this.Center = Center;
            this.Radius = Radius;
        }
    }
}
```

המחלקה הראשית (Game1)

מחלקה זו היא המחלקה הראשית של המשחק והיא זאת שלמעשה מריצה את כל המשחק.
מחלקה זו פועלת כך שישנן כמנ פעולות אשר מזומנות פעם אחת לשם אתחול משתנים וטעינת מידע הנחוץ למשחק ולאחר מכן היא מזמנת את פעולת העדכון 60 פעם בשנייה וכך גם פעולת הציור.
כיוון שאני עובד עם מנגנון החלפת המסכים מטרת מחלקה זו היא רק לדאוג לעדכון המצב הקיים וציורו על המסך. כך למשל, עם הרצת המשחק, בהתחלה מאותחלים כל המשתנים הגלובלים כך שנוכל לגשת אליהם עם הרצת המצבים וכך גם מאותחלים המילונים בהם אני משתמש וישר לאחר מכן מזומן המצב הראשון שהוא – מצב פתיחה. וכעת, כל מה שנותר על מחלקת Game1 לעשות הוא לעדכן את המצב הנתון ולבדוק אם אין מצב חדש שנוצר, במקרה בו יש, המצב הנוכחי יתחלף למצב שזומן.

משתנים:

- האובייקט graphics מסוג GraphicsDeviceManager שאחראי על התיאום על הכרטיס הגרפי.
- האובייקט spriteBatch מסוג SpriteBatch שבעזרתו מייצרים טקסטורות על חלון המשחק. זהו כלי עזר שהתווסף עם ספריות ה-XNA .
- האובייקט Content מסוג ContentManager שבעזרתו טוענים קבצים למשחק (טקסטורות, קבצי XML , פסי קול ועוד..)
- משתנה currentState – מחזיק בתוכו את המצב הנוכחי של המשחק.
- משתנה nextState – מחזיק בתוכו את המצב הבא של המשחק (אם קיים).

פעולות:

- קונסטרקטור Game1 – הפעולה הבונה של המחלקה, נקראת פעם אחת בזמן הפעלת המשחק.
- Initialize – פעולת האתחול של המשחק ונועדה להגדרת מאפיינים בסיסיים של המשחק.
- LoadContent – פעולה המשמשת לייבוא תכנים ומדיה לתוך המשחק וגם מידע הקשור לכרטיס הגרפי.
- Update – מקבלת פרמטר מהטיפוס GameTime המספק נתונים לגבי משך הריצה של המשחק. פעולה זאת מעדכנת את הנתונים של המשחק ומזומנת 60 פעם בשנייה.
- Draw – פעולה זאת מקבלת פרמטר מסוג GameTime ואחראית לציור התוכן של המשחק על גבי המסך ומזומנת בתדירות הגבוהה ביותר שהמחשב מאפשר.
- ChaneState – פעולה זו מקבלת מצב ומעדכנת את משתנה המצב הבא להיות למצב זה.

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

namespace Slugs
{

    public class Game1 : Game
    {
        GraphicsDeviceManager graphics;
```

```
        SpriteBatch spriteBatch;

        private State currentState;
        private State nextState;

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

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

        protected override void Initialize()
        {
            IsMouseVisible = true;

            graphics.PreferredBackBufferWidth = 1000;//600;
            graphics.PreferredBackBufferHeight = 700;//480;
            graphics.ApplyChanges();

            base.Initialize();
        }


        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            General.font = Content.Load<SpriteFont>("FFont");

            General.device = graphics.GraphicsDevice;
            General.sb = spriteBatch;
            General.gp = graphics;
            General.cm = Content;

            Dictionaries.AnimationDicInit();
            Dictionaries.TexturesDicInit();

            General.screenWidth =
General.device.PresentationParameters.BackBufferWidth;
            General.screenHeight =
General.device.PresentationParameters.BackBufferHeight;

            currentState = new MenuState(this);
        }


        protected override void UnloadContent()
        {
        }

        protected override void Update(GameTime gameTime)
        {
            General.Time = (int)gameTime.TotalGameTime.TotalMilliseconds;

            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);

            base.Update(gameTime);
        }


        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);
            spriteBatch.Begin();

            currentState.Draw(gameTime, General.sb);

            spriteBatch.End();

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

מחלקת Slug

מחלקה זו אחראית על שחקן במשחק (יורשת Animation). כאן נמצאות כל התכונות וכל הפעולות הקשורות ופועלות עם השחקן ובעזרתה אנו יוצרים מופעים של שחקנים חדשים.

קצת על שחקן:

כל שחקן הנמצא במשחק נשלט על ידי סוג מקלדת (בוט או מקלדת ידנית) ולו תכונות רבות היוצרות התנהגויות שונות במהלך המשחק. לשחקן האפשרות לנוע בכל מקום במפת המשחק ובכל רגע נתון פועל עליו כוח המשיכה אשר מושך אותו למעלה. כמו כן, השחקן יכול לקפוץ ואף לשאת נשקים ולירות בהם, כגון, בזוקה, תת-מקלע ועוד. בנוסף, קיים לכל שחקן כמות חיים המוגדרת מהתחלת המשחק ומשתנה בהתאם לאינטראקציה שלו עם גופים שונים. לדוגמה, בעת נפילה ממקום גבוה יורד לשחקן חיים בהתאם לגובה. כמו כן, בעת התפוצצות של טיל ליד השחקן יורד לו חיים גם כן בהתאם למרחק הפיצוץ ממנו וכדומה.

פעולות:

- בנאי Slug אשר מאתחל את תכונות השחקן.
- RemoveWeaponShootableAndUpdate – פעולה זו מקבלת ירייה ואחראית להוציא אותה מרישמת היריות שנורו.
- IsOnGround – בודקת אם השחקן על האדמה או לא.
- UpdateHealth – עדכון החיים של השחקן בהתאם למספר שנשלח.
- StartPlaying – פעולה המאתחלת את התכונות של השחקן הנחוצות בהתחלת כל תור.
- UpdateState – פעולות עדכון של השחקן וכל תכונותיו בהתאם למצב המשחק.
- PlayerMovement – פעולה האחראית לתנועת השחקן המשחק.
- Input – אחראית לקלט של המקלדת (בוט או מקלדת ידנית).
- Move – מעדכנת את מיקום השחקן בהתאם לקלט ומצב השחקן.
- Gravity – פעולה המפעילה את כוח המשיכה על השחקן.
- FindTheGround – פעולה המוצאת את מיקום האדמה מתחת לשחקן.
- CheckPlayerCollision – פעולה המקבלת ירייה ובודקת אם יש התנגשות שלה עם השחקן.
- Collision – בדיקת התנגשות בין שני שחקנים.
- פעולת ציור השחקן וכל תכונותיו.
- פעולת ציור הטקסט של זמן המשחק הנותר.

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

using Slugs.Weapons;

namespace Slugs
{
    public class Slug : Animation
    {
        #region Data
        public Heroes hero { get; }
        BaseKeys keyboard;

        public int Health { set;  get; }
```

```
        public int FallHealth { set; get; }

        Vector2 drc;
        float gravitaion;

        public float velocityX { get; set; }
        public float velocityY { get; set; }

        float accelerationY;

        public bool IsAlive { get; set; }
        public bool IsPlaying { get; set; }
        public bool IsShot  { get; set; }

        public Weapon PWeapon { get; set; }
        public float Power { get; set; }
        public float WeaponAngle { get; set; }

        public int PlayTime { get; set; }
        public int PrepTime { get; set; }

        public int gravityActivateTime = 0;
        public int gravityDelay = 500;
        public bool canWalk;

        public int numOfTeam;
        Arrow arrow;

        private Vector2 posOfWeapon = new Vector2(3, -10);

        #endregion

        #region Ctors
        public Slug(Heroes hero, Vector2 position, bool isPlaying, Color color, int
numOfTeam, Texture2D rocketTexture, float rotation, float scale, float layerDepth,
                        Keys left, Keys right, Keys up, Keys down, Keys shift,
Keys Q, Keys E, bool isBot) : base(hero, PlayerState.Stance, position, color,
rotation, scale, layerDepth)
        {
            this.hero = hero;
            Position = position;
            IsAlive = true;
            this.numOfTeam = numOfTeam;
            Health = 100;
            FallHealth = 0;
            IsPlaying = isPlaying;
            Power = 100;
            WeaponAngle = MathHelper.ToRadians(90);

            PWeapon = new
Bazooka(Dictionaries.TextureDictionary[FolderTextures.Graphics]["Bazooka"], new
Vector2(Position.X + 3, Position.Y - 10), WeaponAngle, null, Color.White,
WeaponAngle, new Vector2(15, 36), 0.6f, this.Effects, 0);

            if (!isBot)
                keyboard = new UserKeys(right, left, up, down, shift, Q, E);
            else
                keyboard = new BotKeyboard(this);
            drc = Vector2.Zero;
            gravitaion = 0.5f;

            drc = new Vector2(1, 1);
```

```
            accelerationY = 0.16f;
            velocityX = 0;
            velocityY = 0;

            PlayTime = General.Time + 60000;
            PrepTime = General.Time + 3000;

            arrow = new
Arrow(Dictionaries.TextureDictionary[FolderTextures.Graphics]["RedArrow"], Position,
Color.White, new Vector2(32, 80));

        }
        #endregion

        public void RemoveWeaponShootableAndUpdate(Shootable shootable)
        {
            PWeapon.RemoveShootable(shootable);
        }

        public bool IsOnGround()
        {
            if (General.terrainMask[(int)this.Position.X, (int)this.Position.Y] ==
1)
                return true;
            return false;
        }

        public void UpdateHealth(int damageTaken)
        {
            if (damageTaken >= Health)
                damageTaken = Health;

            Health -= damageTaken;
            General.currentTeamHealth[numOfTeam] -= damageTaken;
        }

        public void StartPlaying()
        {
            IsPlaying = true;
            IsShot = false;
            PlayTime = General.Time + 60000;
            PrepTime = General.Time + 3000;

            PWeapon.Reset();
        }

        public void UpdateState()
        {
            keyboard.Update();
            PWeapon.Update(Position + posOfWeapon, WeaponAngle, this.Effects);
            if (this.Health <= 0)
            {
                this.IsAlive = false;
                if (IsPlaying && !IsShot)
                    General.NextPlayer();
            }

            if (IsPlaying)
            {
                if (General.Time > PrepTime)
                {
                    Input();
```

```
                    arrow.PostUpdate(Position);

                    if (General.Time >= PlayTime && IsPlaying && !IsShot)
                    {
                        arrow.PostUpdate(Position);
                        General.NextPlayer();
                    }
                }
                else
                {
                    arrow.Update();
                }
            }
            else
                arrow.PostUpdate(Position);

            PlayerMovement();

            if (!IsPlaying)
            {
                SpriteSheetAnimation =
Dictionaries.PlayerAnimationDictionary[hero][PlayerState.Stance];
            }
        }

        public void PlayerMovement()
        {
            Move();
            base.Update();
        }

private void Input()
        {
            if (keyboard.LeftPressed())
            {
                if (Effects == SpriteEffects.None)
                    WeaponAngle = -WeaponAngle;

                if ((SpriteSheetAnimation !=
Dictionaries.PlayerAnimationDictionary[hero][PlayerState.Walk] || Effects ==
SpriteEffects.None))
                {
                    if (IsOnGround())
                    {
                        posOfWeapon = new Vector2(3, -10);
                        CurrentIndex = 0;
                        FrameDelay = 0;
                    }
                    Effects = SpriteEffects.FlipHorizontally;
                }

                if (canWalk)
                    velocityX -= 1;

                if (keyboard.SpacePressed())
                {
                    posOfWeapon = new Vector2(2, -22);
                    SpriteSheetAnimation =
Dictionaries.PlayerAnimationDictionary[hero][PlayerState.Jump];
                }
                else if (IsOnGround())
                {
```

```
                    posOfWeapon = new Vector2(3, -10);
                    SpriteSheetAnimation =
Dictionaries.PlayerAnimationDictionary[hero][PlayerState.Walk];
                }
            }
            else if (keyboard.RightPressed())
            {
                if (Effects == SpriteEffects.FlipHorizontally)
                    WeaponAngle = -WeaponAngle;

                if ((SpriteSheetAnimation !=
Dictionaries.PlayerAnimationDictionary[hero][PlayerState.Walk] || Effects ==
SpriteEffects.FlipHorizontally))
                {
                    if (IsOnGround())
                    {
                        posOfWeapon = new Vector2(3, -10);
                        CurrentIndex = 0;
                        FrameDelay = 0;
                    }
                    Effects = SpriteEffects.None;
                }

                if (canWalk)
                    velocityX += 1;

                if (keyboard.SpacePressed())
                {
                    posOfWeapon = new Vector2(2, -22);
                    SpriteSheetAnimation =
Dictionaries.PlayerAnimationDictionary[hero][PlayerState.Jump];
                }
                else if (IsOnGround())
                {
                    posOfWeapon = new Vector2(3, -10);
                    SpriteSheetAnimation =
Dictionaries.PlayerAnimationDictionary[hero][PlayerState.Walk];
                }

            }
            else if (keyboard.SpacePressed() && IsOnGround())
            {
                posOfWeapon = new Vector2(2, -22);
                SpriteSheetAnimation =
Dictionaries.PlayerAnimationDictionary[hero][PlayerState.Jump];
                CurrentIndex = 0;
                FrameDelay = 0;
            }
            else if (SpriteSheetAnimation !=
Dictionaries.PlayerAnimationDictionary[hero][PlayerState.Stance] && IsOnGround())
            {
                posOfWeapon = new Vector2(3, -10);
                SpriteSheetAnimation =
Dictionaries.PlayerAnimationDictionary[hero][PlayerState.Stance];
                CurrentIndex = 0;
                FrameDelay = 0;
            }

            if (keyboard.QPressed())
            {
                WeaponAngle -= 0.01f;
            }
```

```
            if (keyboard.EPressed())
            {
                WeaponAngle += 0.01f;
            }

            if (Effects == SpriteEffects.None)
            {
                if (WeaponAngle > MathHelper.ToRadians(180))
                    WeaponAngle = MathHelper.ToRadians(180);
                else if (WeaponAngle < 0)
                    WeaponAngle = 0;
            }
            else
            {
                if (WeaponAngle > 0)
                    WeaponAngle = 0;
                else if (WeaponAngle < MathHelper.ToRadians(-180))
                    WeaponAngle = MathHelper.ToRadians(-180);
            }

            if (keyboard.DownPressed())
            {
                Power -= 1;
            }

            if (keyboard.UpPressed())
            {
                Power += 1;
            }

            if (Power > 1000)
            {
                Power = 1000;
            }
            if (Power < 0)
            {
                Power = 0;
            }

            if (keyboard.OnePressed())
                PWeapon = new
Bazooka(Dictionaries.TextureDictionary[FolderTextures.Graphics]["Bazooka"], new
Vector2(Position.X + 3, Position.Y - 10), PWeapon.Angle, null, Color.White,
PWeapon.Angle, new Vector2(15, 36), 0.6f, this.Effects, 0);
            else if (keyboard.TwoPressed())
                PWeapon = new
Uzi(Dictionaries.TextureDictionary[FolderTextures.Graphics]["Uzi"], new
Vector2(Position.X + 3, Position.Y - 10), PWeapon.Angle, null, Color.White,
PWeapon.Angle, new Vector2(21, 30), 0.6f, this.Effects, 0);
            else if (keyboard.ThreePressed())
                PWeapon = new
Shotgun(Dictionaries.TextureDictionary[FolderTextures.Graphics]["Shotgun"], new
Vector2(Position.X + 3, Position.Y - 10), PWeapon.Angle, null, Color.White,
PWeapon.Angle, new Vector2(10, 39), 0.6f, this.Effects, 0);


            if (keyboard.SpacePressed() && IsOnGround())
            {
                velocityY -= 5f;
            }
```

```
            if (keyboard.EnterPressed() && !IsShot)
            {
                IsShot = true;
                PWeapon.ShootWeapon(Position + posOfWeapon, WeaponAngle, Power);
            }
        }

public void Move()
        {
            float startPositionY = Position.Y;
            int groundY = FindTheGround();

            if (groundY == (int)(Position.Y - Texture.Height))
                Position = new Vector2(Position.X + velocityX, Position.Y + 1);

            Gravity(groundY);

            float endPositionY = Position.Y;

            if (startPositionY < endPositionY)
                velocityX *= 2;

            else if (startPositionY > endPositionY)
                velocityX *= 0.1f;

            if (velocityY >= 6f)
                FallHealth += 1;



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

            if (Position.Y == groundY)
            {
                velocityX = 0;
                UpdateHealth(FallHealth);
                FallHealth = 0;
            }

        }

        private void Gravity(int groundY)
        {
            if (this.Position.Y + velocityY < groundY)
            {
                if (General.Time - gravityActivateTime >= gravityDelay)
                    canWalk = false;
                velocityY += accelerationY;
            }
            else
            {
                canWalk = true;
                Position = new Vector2(Position.X, groundY);
                velocityY = 0;
            }
        }

        private int FindTheGround()
        {
            if (Position.Y - Texture.Height > 0 && Position.X + velocityX <
General.screenWidth && Position.X + velocityX > 0)
```

```
                for (int i = (int)Position.Y - Texture.Height; i <
General.screenHeight; i++)
                {
                    if (General.terrainMask[(int)(Position.X + velocityX), i] == 1)
                    {
                        if (i == (int)(Position.Y - Texture.Height))
                        {
                            velocityX = 0;
                            velocityY = -velocityY;

                            if (General.terrainMask[(int)(Position.X + velocityX),
(int)Position.Y] == 1)
                                return (int)Position.Y;

                            return (int)Position.Y + 1;
                        }
                        return i;
                    }
                }
            velocityX = 0;
            return (int)Position.Y;
        }

        public Vector2 CheckPlayerCollision(Shootable shootable)
        {
            Vector2 playerCollisionPoint = new Vector2(-1, -1);

            if (this.IsAlive)
            {
                int xPos = (int)this.Position.X;
                int yPos = (int)this.Position.Y;

                foreach (Circle c in
this.SpriteSheetAnimation.AllPagesCircle[CurrentIndex])
                {
                    Vector2 center1 = Vector2.Transform(c.Center,
Matrix.CreateRotationZ(Rotation));
                    Vector2 v1 = Position + center1 * Scale;

                    Vector2 rocketNose = shootable.GetShootableNose();

                    if ((v1 - rocketNose).Length() < (c.Radius * Scale))
                        return rocketNose;

                    if ((v1 - rocketNose - shootable.Direction / 2).Length() <
(c.Radius * Scale))
                        return rocketNose;

                    if ((v1 - rocketNose - shootable.Direction).Length() < (c.Radius
* Scale))
                        return rocketNose;
                }
            }

            return new Vector2(-1, -1);
        }

        public bool Collision(Slug slug)
        {
            if (this.SpriteSheetAnimation.AllPagesCircle != null)
            {
```

```
                foreach(Circle c in
this.SpriteSheetAnimation.AllPagesCircle[CurrentIndex])
                {
                    Vector2 center1 = Vector2.Transform(c.Center,
Matrix.CreateRotationZ(Rotation));
                    Vector2 v1 = Position + center1 * Scale;
                    foreach (Circle c2 in
slug.SpriteSheetAnimation.AllPagesCircle[slug.CurrentIndex])
                    {
                        Vector2 center2 = Vector2.Transform(c2.Center,
Matrix.CreateRotationZ(slug.Rotation));
                        Vector2 v2 = slug.Position + center2 * slug.Scale;

                        if ((v1 - v2).Length() < (c.Radius * Scale + c2.Radius *
slug.Scale))
                            return true;
                    }
                }
            }
            return false;
        }

        public override void Draw()
        {
            base.Draw();
            if (IsPlaying)
            {
                PWeapon.Draw();

                if (General.Time <= PrepTime)
                    arrow.Draw();
            }
            General.sb.DrawString(General.font, Health.ToString(), new
Vector2(Position.X - 10, Position.Y - 55), color);
        }

        public void DrawPlayTime()
        {
            if (General.Time > PrepTime)
                General.sb.DrawString(General.font, "Time Left:  " + (PlayTime -
General.Time) / 1000 / 60 + " : " + (PlayTime - General.Time) / 1000, new
Vector2(20, 70), color);
            else
                General.sb.DrawString(General.font, "Prep. Time: " + (PrepTime -
General.Time) / 1000 / 60 + " : " + (PrepTime - General.Time) / 1000, new
Vector2(20, 70), color);
        }

    }
}
```

מנגנון הפיצוץ:

בפרויקט שלי היה חשוב לי לנסות ולבצע פיצוץ כמה שיותר אמיתי וכך גם שישפיע על אדמת המשחק באופן שונה בכל פעם ועבור כל ירייה.
על מנת לבצע מטרה זו חקרתי את נושא הפיצוצים במשחקים שונים למדתי שניתן לממש זאת על ידי יצירת חלקיקים אשר לכל אחד מהם התנהגות שונה לחלוטין, כלומר, לחלקיק יהיו התכונות הבאות: זמן יצירה, אורך חיים, מיקום התחלתי, מיקום נוכחי, תאוצה וכיוון. על ידי תכונות אלו ניתן ליצור פיצוצים שונים כאשר משנים את אורך החיים, תאוצת החלקיקים וכו'.

מחלקת Particle

מחלקה זו יורשת את מחלקת Drawable ומכילה בתוכה את הפרטים של חלקיק מתוך פיצוץ מסויים של טיל בקרקע או בשחקן.

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

namespace Slugs
{
    public class Particle : Drawable
    {
        public float BirthTime { get; set; }
        public float MaxAge { get; set; }
        public Vector2 OrginalPosition { get; set; }
        public Vector2 Accelaration { get; set; }
        public Vector2 Direction { get; set; }


        public Particle(float birthTime, float maxAge, Vector2 originalPosition,
            Vector2 accelaration, Vector2 direction, Texture2D texture, Vector2
position, Rectangle? sourceRectangle,
            Color color, float rotation, Vector2 origin, float scale, SpriteEffects
effects, float layerDepth)

            : base(texture, position, sourceRectangle, color, rotation, origin,
scale, effects, layerDepth)
        {
            BirthTime = birthTime;
            MaxAge = maxAge;
            OrginalPosition = originalPosition;
            Accelaration = accelaration;
            Direction = direction;
        }
    }
}
```

מחלקת Explosion
מחלקה סטטית האחראית על פיצוץ של יריה עם הקרקע או השחקן.

פעולות:

- Create – אתחול תכונות המחלקה.
- UpdateParticles – עדכון החלקיקים בהתאם לתכונותיהם.
- AddExplosion – פעולה המקבלת את מיקום הפיצוץ, מספר החלקיקים, גודל החלקיקים ואורך החיים ויוצרת את מספר החלקיקים בהתאם לקלט – יוצרת פיצוץ.
- AddExplosionParticle – הוספת חלקיק לרשימת החלקיקים בהתאם לקלט.
- פעולת ציור הפיצוץ.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Linq;
using System.Collections.Generic;
namespace Slugs
{
    static class Explosion
    {
        public static Texture2D Texture;
        public static Color[,] explosionColorArray;
        public static List<Particle> particleList;

        public static void Create(Texture2D texture)
        {
            Texture = texture;
            explosionColorArray = General.TextureTo2DArray(texture);
            particleList = new List<Particle>();
        }
        public static void UpdateParticles(GameTime gameTime)
        {
            float now = (float)gameTime.TotalGameTime.TotalMilliseconds;
            for (int i = particleList.Count - 1; i >= 0; i--)
            {
                Particle particle = particleList[i];
                float timeAlive = now - particle.BirthTime;

                if (timeAlive > particle.MaxAge)
                {
                    particleList.RemoveAt(i);
                }
                else
                {
                    // realtive age
                    float relAge = timeAlive / particle.MaxAge;

                    // the position according to a physics formula
                    particle.Position = 0.5f * particle.Accelaration * relAge *
relAge + particle.Direction * relAge + particle.OrginalPosition;

                    // inverted age
                    float invAge = 1.0f - relAge;

                    // colors the particle according to the age of it - in te end it
is gone (0)
                    particle.color = new Color(new Vector4(invAge, invAge, invAge,
invAge));
```

```
                    Vector2 positionFromCenter = particle.Position -
particle.OrginalPosition;
                    float distance = positionFromCenter.Length();
                    particle.Scale = (50.0f + distance) / 200.0f;

                    particleList[i] = particle;
                }
            }
        }
        public static void AddExplosion(Vector2 explosionPos, int numberOfParticles,
float size, float maxAge, GameTime gameTime)
        {
            for (int i = 0; i < numberOfParticles; i++)
                AddExplosionParticle(explosionPos, size, maxAge, gameTime);

            // Add a crater on the foreground by randomizing the rotation of the
texture and getting the matrix of it by the rotation, scale and position of the
explosion.
            float rotation = (float)General.randomizer.Next(10);
            Matrix mat = Matrix.CreateTranslation(-Texture.Width / 2, -
Texture.Height / 2, 0) * Matrix.CreateRotationZ(rotation) * Matrix.CreateScale(size
/ (float)Texture.Width * 2.0f) * Matrix.CreateTranslation(explosionPos.X,
explosionPos.Y, 0);
            Foreground.AddCrater(explosionColorArray, mat);
        }
        public static void AddExplosionParticle(Vector2 explosionPos, float
explosionSize, float maxAge, GameTime gameTime)
        {
            float BirthTime = (float)gameTime.TotalGameTime.TotalMilliseconds;

            float particleDistance = (float)General.randomizer.NextDouble() *
explosionSize;
            Vector2 displacement = new Vector2(particleDistance, 0);
            float angle = MathHelper.ToRadians(General.randomizer.Next(360));
            displacement = Vector2.Transform(displacement,
Matrix.CreateRotationZ(angle));

            Vector2 direction = displacement * 2.0f;
            Vector2 accelaration = -direction;

            // TODO: change the rotation to i from 0!!
            Particle particle = new Particle(BirthTime, maxAge, explosionPos,
accelaration, direction, Texture, explosionPos, null, Color.White, 0, new
Vector2(256, 256), 0.25f, SpriteEffects.None, 1);

            particleList.Add(particle);
        }
        public static void Draw()
        {

            for (int i = 0; i < particleList.Count; i++)
            {
                Particle particle = particleList[i];
                General.sb.Draw(Texture, particle.Position,
particle.SourceRectangle, particle.color, i, particle.Origin, particle.Scale,
particle.Effects, 0);
            }

        }
    }
}
```

מנגנון "לפי פיקסל" – מנגנון ההתנגשות השני

בפרויקט זה בבדיקת ההתנגשות של פיצוץ באדמת המשחק פעלתי בבדיקה לפי פיקסל על מנת שהבדיקה של המגע בין הפיצוץ לאדמה או לשחקן תהיה מדוייקת, דבר החשוב ספציפית למשחק זה כיוון שאני עובד עם הרס של האדמה בהתאם למיקום הפיצוץ וגודלו.

בפרויקט זה היה חשוב לי שהפיצוצים במשחק יהיו כמה שיותר שונים כדי שלשחקן תהיה תחושה שהפיצוץ הוא דינאמי ורנדומלי ולא תמיד יהיו אותן תוצאות.

מנגנון זה עובד כך שהוא עובר על מטריצת הפיקסלים של כל טקסטורה ועושה התאמה של שניהם ביחס למסך ולאחר מכן בודק בדיוק איפה מתרחשת הנגיעה של הטקסטורות.

מחלקת Foreground

מחלקה סטטית אשר מטרת מחלקה זו היא לדאוג לכל הקשור לאדמת המשחק שעליה נמצאים השחקנים.
אובייקט הForeground משתמש במנגנון "לפי פיקסל" ולכן אחת התכונות שלו הינה מערך הצבע של טקסטורת האדמה.

פעולות:

- בנאי Foreground – הבנאי מזמן את הפעולה העוברת על מערך הקרקע של הדמויות ומתאים את הטקסטורה של האדמה לקרקע.
- בדיקת התנגשות של יריה עם הקרקע – הפעולה מקבלת יריה הנורית על ידי שחקן ובודקת אם קיימת התנגשות בינהם.
- יצירת מכתש – במקרה של פגיעת יריה בקרקע, פעולה זו תזומן ויווצר מכתש במיקום על הקרקע בו פגעה היריה.

```
    public static class Foreground
    {
        public static Texture2D MaskTexture;
        static Texture2D groundTexture;
        static Color color;
        static Rectangle destinationRectangle;
        static Color[] foregroundColors;
        public static void Create(Texture2D maskTexture, Texture2D groundTex,
Rectangle destinationRec, Color texColor)
        {
            //MaskTexture = maskTexture;
            MaskTexture = new Texture2D(General.gp.GraphicsDevice,
General.screenWidth, General.screenHeight);
            groundTexture = groundTex;
            destinationRectangle = destinationRec;
            color = texColor;
            CreateForeground();
        }
        public static void CreateForeground()
        {
            // Get the a 2D array of the ground texture
            Color[,] groundColors = General.TextureTo2DArray(groundTexture);
            foregroundColors = new Color[General.screenWidth *
General.screenHeight];
            for (int x = 0; x < General.screenWidth; x++)
            {
                for (int y = 0; y < General.screenHeight; y++)
```

```
                {
                    // Paint the terrain mask with the ground texture.
                    if (General.terrainMask[x, y] == 1)
                        foregroundColors[x + y * General.screenWidth] =
groundColors[x % groundTexture.Width, y % groundTexture.Height];
                    else
                        foregroundColors[x + y * General.screenWidth] =
Color.Transparent;
                }
            }
            MaskTexture.SetData(foregroundColors);
        }
        public static Vector2 CheckTerrainCollision(Shootable shootable)
        {
            Vector2 pos = shootable.GetShootableNose();
            if ((pos.Y - 2 >= 0 && pos.Y + 2 < General.screenHeight) && (pos.X - 2
>= 0 && pos.X + 2 < General.screenWidth))
                for (int i = (int)(pos.X - 2); i < pos.X + 2; i++)
                {
                    for (int j = (int)(pos.Y - 2); j < pos.Y + 2; j++)
                    {
                        if (General.terrainMask[i, j] == 1)
                            return pos;
                    }
                }
            return new Vector2(-1, -1);
        }
        public static void AddCrater(Color[,] tex, Matrix mat)
        {
            int width = tex.GetLength(0);
            int height = tex.GetLength(1);

            for (int x = 0; x < width; x++)
            {
                for (int y = 0; y < height; y++)
                {
                    if (tex[x, y].R > 10)
                    {
                        Vector2 imagePos = new Vector2(x, y);
                        Vector2 screenPos = Vector2.Transform(imagePos, mat);

                        int screenX = (int)screenPos.X;
                        int screenY = (int)screenPos.Y;

                        if ((screenX) > 0 && (screenX < General.screenWidth) &&
screenY > 0 && screenY < General.screenHeight)
                        {
                            General.terrainMask[screenX, screenY] = 0;
                            foregroundColors[screenX + screenY *
General.screenWidth] = Color.Transparent;
                        }
                    }
                }
            }
            // Update the Mask texture.
            MaskTexture.SetData(foregroundColors);
        }
        public static void Draw()
        {
            General.sb.Draw(MaskTexture, destinationRectangle, color);
        }
    }
```

מחלקת BotKeyboard

מחלקה זו יורשת את מחלקת BaseKeys ואחראית על הכפתורים השייכים לבוט במשחק.
המחלקה מממשת את הפעולות שהוגדרו מחלקת BaseKeys.
למעשה זוהי המחלקה שמממשת את הבוט כולו. היא פועלת כך שבפעולת העדכון יש את כל אלגוריתם הבינה של הבוט ובהתאם ולמשתנים שהגדרתי אני מעדכן את מצב המקלדת כאילו לחץ על הכפתורים ובמחלקת ה-Slug על הבדיקות בהתאם מתבצעות כבר.

```
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;

namespace Slugs
{
    class BotKeyboard : BaseKeys
    {
        Slug bot;
        Vector2 direction;
        Vector2 weaponRotation;
        float power;
        Slug target;
        float targetDist;
        bool isShot;

        public BotKeyboard(Slug bot) : base()
        {
            direction = Vector2.Zero;

            this.bot = bot;
            this.target = null;
            this.targetDist = 0;
            isShot = false;

        }
        public override Boolean RightPressed()
        {
            return (direction.X > 0);
        }
        public override Boolean LeftPressed()
        {
            return (direction.X < 0);
        }
        public override Boolean UpPressed()
        {
            return (power > 0);
        }
        public override Boolean DownPressed()
        {
            return (power < 0);
        }
        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 SpacePressed()
        {
            return false;
        }
        public override bool QPressed()
        {
            return (weaponRotation.X < 0);
        }
        public override bool EPressed()
        {
            return (weaponRotation.X > 0);
        }
        public override bool QReleased()
        {
            return false;
        }
        public override bool EReleased()
        {
            return false;
        }

        public override bool EnterPressed()
        {
            if (isShot)
            {
                return true;
            }
            return false;
        }

        public override bool EnterReleased()
        {
            return false;
        }

        public override bool SpaceReleased()
        {
            return false;
        }

        public override void Update()
        {
            if (bot.IsPlaying)
            {
                FindClosestSlug();

                if (target != null)
                {
                    Vector2 shootVector = target.Position - bot.Position;
                    float angleFromVector = (float)Math.Atan2(shootVector.X, -
shootVector.Y);
                    if (bot.Position.X > target.Position.X)
                    {
```

```
                        if (bot.Effects == SpriteEffects.None)
                        {
                            bot.Effects = SpriteEffects.FlipHorizontally;
                            angleFromVector = -angleFromVector;
                        }

                    }
                    else if (bot.Position.X < target.Position.X)
                    {
                        if (bot.Effects == SpriteEffects.FlipHorizontally)
                        {
                            bot.Effects = SpriteEffects.None;
                            angleFromVector = -angleFromVector;
                        }
                }

                    bot.WeaponAngle = angleFromVector;
                    isShot = true;
                }
                else
                    isShot = false;
            }
        }
        public void FindClosestSlug()
        {
            float minDist = float.MaxValue;
            float currDist;
            for (int i = 0; i < General.players.Length; i++)
            {
                if (General.players[i].numOfTeam != bot.numOfTeam &&
General.players[i].IsAlive)
                {
                    currDist = Math.Abs((bot.Position -
General.players[i].Position).Length());
                    if (minDist > currDist)
                    {
                        minDist = currDist;
                        target = General.players[i];
                        targetDist = minDist;
                    }
                }
            }
        }
        public override bool OnePressed()
        {
            return false;
        }
        public override bool TwoPressed()
        {
            return false;
        }
        public override bool ThreePressed()
        {
            return false;
        }
    }
}
```