פרויקט גמר י"ג הנדסת תכנה

משחק ארקייד פלטפורמה דו ממדי עם בינה מלאכותית

פיתוח: בן-אל מוסיוב

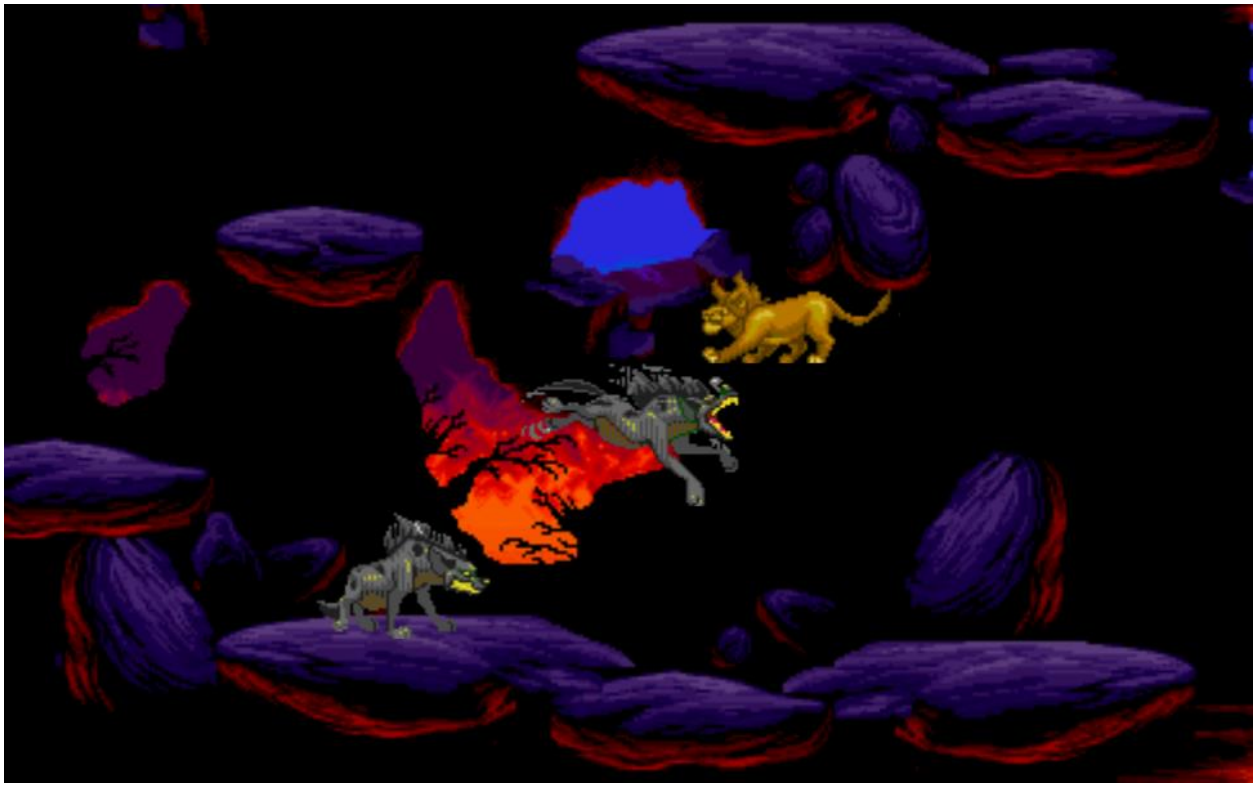
תעודת זהות: 318653300

מנחה: יהודה אור

מכללת הכפר הירוק

תאריך: 24/05/2016

תוכן עניינים



מבוא

נושא העבודה

תכנות משחק ארקייד דו ממדי עם בינה מלאכותית בסביבות VISUAL STUDIO 2013 ו - XNA 4.0 תוך כדי יישום עקרונות הפיזיקה.

המשחק שלי הוא מעין גרסה של משחק הפלטפורמות המקורי המפורסם משנת 1994, מלך האריות - Lion King. במשחק המשתמש שולט בדמות ומטרתו להרוג את המפלצות השונות. כאשר המפלצות מרגישות "מאוימות" הן מתחילות לתקוף (חלקן) ולנוע לכיוון המטרה שלהם במקרה הזה השחקן שתקף אותם.

היעד הוא הבוס בשלב השני והאחרון, הלא הוא סקאר, אשר אותו סימבה הגיבור צריך להביס על ידי הפלתו מן הצוק.

מטרת הפרויקט

המטרה העיקרית שלי בפרויקט הייתה ללמוד ולהעשיר את הידע שלי בתכנות, באלגוריתמיקה ולשלב את הידע שלי במתמטיקה ופיזיקה על מנת ליצור משחק הדומה ככל היותר למציאות.

על מנת לשמור על נוחות ותחזוקת הקוד שהשתמשתי בOOP – Object Oriented Programming חילקתי את כל הפרויקט למבנים פשוטים שישמשו להורשה ופולימורפיזם.

בפרויקט זה חקרתי על שילוב פיזיקה במשחק דו ממד ושילבתי בו את הידע שרכשתי במהלך השנה בבית ספר – כגון מציאת מיקום בתנועה שוות תאוצה.

למדתי על שימוש במטריצות, בקבצי XML על מנת לשמור מידע קבוע, טעינתו במשחק ושימוש בנתונים הכתובים שם.

למדתי מטעויות וכישלונות, ולאחר גרסאות וניסיונות רבים הגעתי לתוצאה המוצגת בפרויקט זה.

למדתי הרבה מהפרויקט, העשרתי והרחבתי את ידיעותיי בתחום התכנות בכלל ובפיתוח משחקים בפרט. אני שמח ומרוצה מביצוע הפרויקט בהצלחה, ומכוון ללמוד ולהתקדם עוד בתחום התכנות.

מבוא לתכנות

מבוא לתכנות ב#C

#C הינה שפת תכנות מונחת עצמים שפותחה על ידי מיקרוסופט כחלק מיוזמת הDOT NET שלה. תחביר השפה דומה לJAVA ו++C.

בשפת #C ממומשים העקרונות הכלליים של התכנות מונחה העצמים: כימוס ,הורשה ופולימורפיזם. בשונה מ ++C ובדומה לJava ההורשה שמאפשרת #C יכולה להיות רק מטיפוס בסיס, ואין תמיכה בהורשה מרובה. לעומת זאת ישנה אפשרות לממש מספר ממשקים.

שפת #C משמשת בטכנולוגיות רבות:

- פיתוח אפליקציות Console
- פיתוח תוכנות בעלי ממשקים גרפים בעזרת Win forms
- פיתוח אתרי אינטרנט דינאמיים באמצעות asp.net.
- שימוש בSilverlight

שפת #C היא שפת Managed Code כלומר המתכנת אינו אחראי יותר על שחרור זיכרון אלה סביבת ההרצה היא זאת שדואגת לשחרור זיכרון של אובייקטים שאינם בשימוש.

קוד #C מהודר לשפת ביניים הנקראת Common Intermediate Language - CIL ובזמן ההרצאה הוא מקומפל לשפת מכונה.

תכנות מונחה עצמים

תבנית המחשבה 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 תחזוקה רבה ואיטיות ומורכבות בפיתוח ככל שהתקדם, וכך, בניגוד למחירי החומרה שכל הזמן ירדו, מחירי התוכנות דווקא האמירו. לפיכך היה צורך לחפש דרך חדשה שתפשט ותקל על עבודת התכנות.

מרכיב בסיסי בתכנות מונחה-עצמים הוא המחלקה. מחלקה היא מבנה מופשט בעל תכונות המגדירות ומאפיינות את המחלקה, ופעולות שהן פונקציות ייחודיות למחלקה. בחלק משפות התכנות קיימים גם אירועים אשר מוזנקים בהקשרים שונים, למשל בתחילתו או בסיומו של הליך או כתגובה לקלט מהמשתמש.

הורשה

באמצעות הורשה ניתן להגדיר מחלקה חדשה על בסיס מחלקה קיימת. למחלקה החדשה אותן תכונות ופעולות של המחלקה שהיא ירשה ובנוסף ניתן להגדיר פעולות ותכונות חדשות למחלקה החדשה ובכך להרחיב את המחלקה שממנה ירשה.

לדוגמא על בסיס המחלקה עוגה אפשר להגדיר את המחלקות עוגת שוקולד, עוגת גבינה, עוגת ביסקוויטים וכדומה, כאשר לכל עוגה תהיה טעם משלה תוספות משלה וכו'.

פולימורפיזם

היכולת לתת משמעויות שונות לפעולה בהקשרים שונים, וכך להתייחס לקבוצה של עצמים שונים באופן אחיד. המשמעות של פעולה נקבעת בהקשר של כל אחת מהמחלקות שבהן מוגדרת פעולה זו, ובהתאם לעצם שבו עוסקת הפעולה.

<u>מושגים הרווחים ברב צורתיות, הם:</u>

- **העמסת פרמטרים** ,שבו ניתן ליצור כמה פונקציות בעלות שם זהה אך בעלות פרמטרים שונים. דבר זה שימושי לריכוז כל הפונקציות תחת אותה קורת גג ,במקום לכתוב פונקציות רבות בעלות שם שונה שמבצעות למעשה את אותה פעולה מבחינה לוגית.
- **העמסת אופרטורים** ,שבו אופרטור מסוים שומר על צורתו הלוגית אך מקבל מכניזם פנימי שונה בתוך אובייקט. למשל אופרטור + שנועד לחבר מספרים ,במחלקה של מחרוזות מקבל יכולת שרשור ,ובמחלקה של מערכים יכול לקבל יכולת לחבר בין שני מערכים.
- **דריסה** (Override) שבה ניתן להחליף פונקציה של מחלקת בסיס בפונקציה שונה במחלקה הנגזרת ממנה.

פונקציה

פעולה/פונקציה היא רצף של פקודות המאוגדות יחד על מנת לבצע מטרה אחת מוגדרת.

פעולות יכולות להחזיר ערך מסוג מסוים (bool, int, float וכו') או לבצע פעולה ללא החזרת ערך(void).

שימוש בפונקציות משפר את מבנה התוכנית, את קריאות הקוד ואת מידת הגמישות של התוכנית לביצוע שינויים. עובדה זו מאפשרת להפחית במידה משמעותית את עלויות הפיתוח והתחזוקה של תוכנה.

Interface

משמש להפשטה של מחלקות התוכנה, ומגדיר את המשתנים והפונקציות שעל מחלקה לממש כדי להיות שייכת אליו. כאשר מחלקה מממשת את כל הפונקציות המוגדרות בממשק ניתן ליצור מופע שלה. שימוש בממשקים הוא נוהג של כתיבה נכונה בהנדסת תוכנה ,כי בשיטה זו מתבצעת הפרדה בין המימוש בפועל לבין הדרישות שמאופיינות בממשק.

פיתוח ב – XNA

XNA הוא אוסף של כלים המיועד לפיתוח משחקי מחשב שהושק על ידי מיקרוסופט. אוסף הכלים של XNA הוכרז ב24 למרץ 2004 בכנס מפתחי משחקים בסן חוזה בקליפורניה ובדצמבר 2007 יצא XNA Game Studio 2. שנה לאחר מכן יצא גרסה 3 ובשנת 2010 גרסה 4 עם אפשרות לפיתוח ל7 Windows phone.

ספריית XNA פותחה על מנת לאפשר פיתוח משחקים בנוחות ויעילות.

קצת על מלך האריות...

מלך האריות : The Lion King)באנגלית (הוא סרט הרפתקאות מוזיקלי אמריקאי בהנפשה קלאסית של וולט דיסני מ .1994-הסרט בוים על ידי רוג'ר אלרס ורוב מינקוף ונחשב לאחד מסרטי הרנסאנס של דיסני.

הסרט מספר את סיפורו של סימבה ,גור אריות שעתיד לרשת את תפקיד המלך מאביו מופאסה. לאחר שדודו של סימבה, סקאר, רוצח את מופאסה כדי לזכות בשליטה על להקת האריות, סימבה טועה לחשוב שמות אביו היה באחריותו, ולכן מחליט להגלות את עצמו מהלהקה. זמן קצר לאחר עזיבת הלהקה, סימבה מגיע לאפיסת כוחות, ואוכלי נבלות מנסים לאכול אותו .סוריקטה וחזיר יבלות אשר נקראים טימון ופומבה נקלעים למקום, מצילים אותו ומגדלים אותו. מאחר שסימבה לא נמצא בממלכת האריות על מנת להנהיגה, סקאר הופך למלך וכורת ברית עם צבועים .לאחר תקופה ארוכה בה סימבה לא פגש אף אחד מלהקתו הקודמת מוצאת אותו נלה ,לביאה שהייתה חברת ילדות של סימבה. נלה משכנעת אותו לחזור לממלכה כדי להביס את סקאר, אשר תפס את השלטון. בעזרת חבריו חוזר סימבה לממלכה, הורג את סקאר ומגרש את הצבועים. עלילת הסרט הושפעה, בין היתר, מהסיפורים התנ"כיים אודות יוסף ומשה ומהמחזה "המלט" פרי עטו של שייקספיר.

פיזיקה

וקטורים:

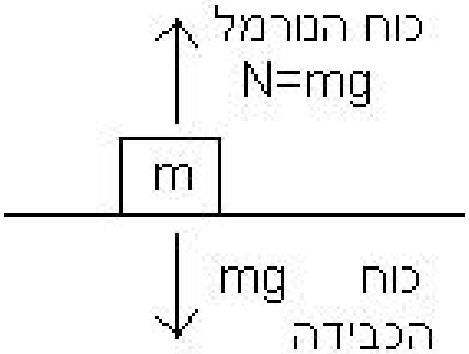
וקטור הוא גודל פיזיקלי שמייצג גודל וכיוון. במשחק שימש אותי בעיקר לייצוג מהירות ומיקום.
סקלר מייצג רק גודל קבוע ללא כיוון.

מיקום משתנה בתנועה שוות תאוצה:

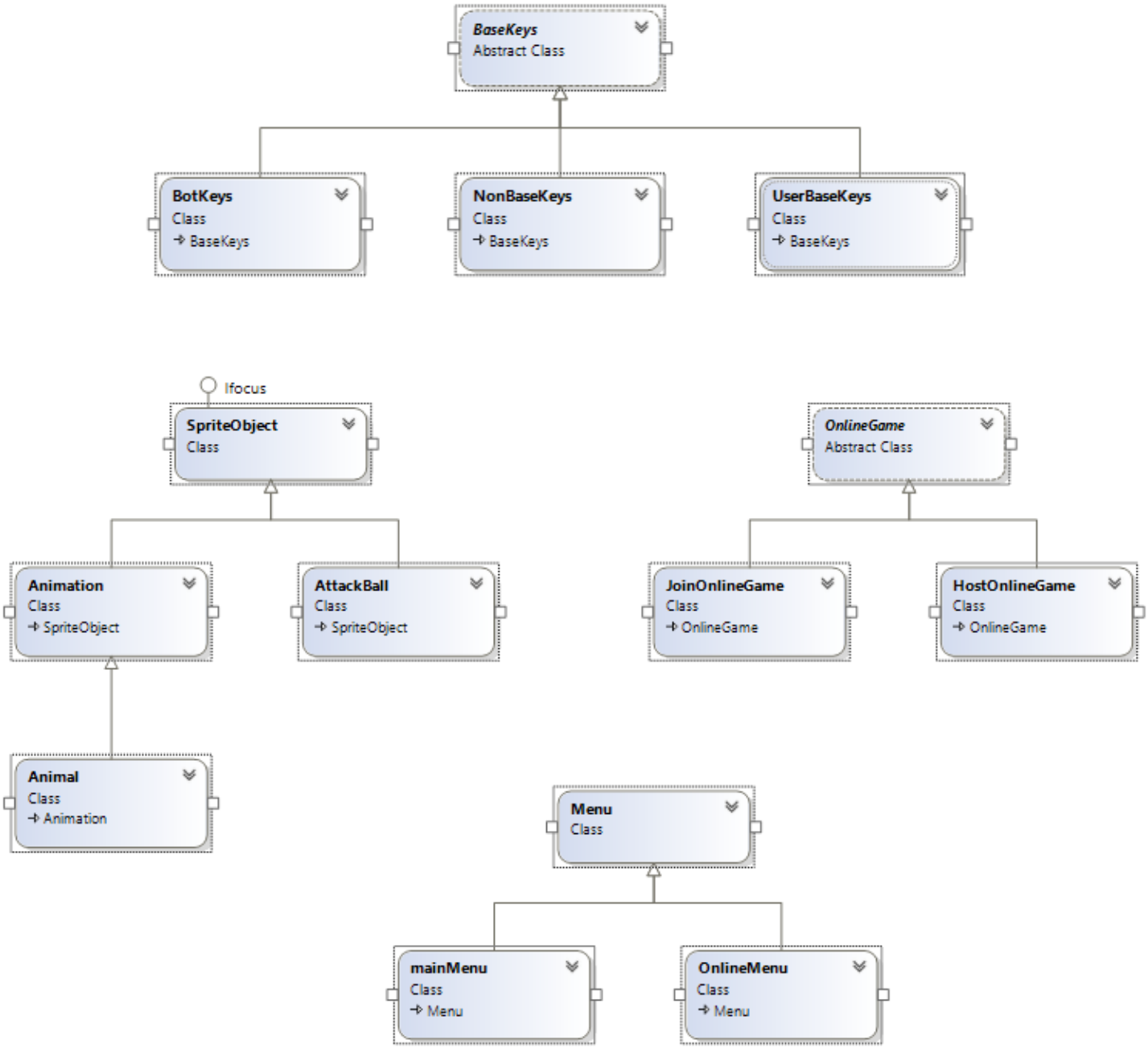
$$x = x_0 + v_0 t + \frac{1}{2} a t^2$$

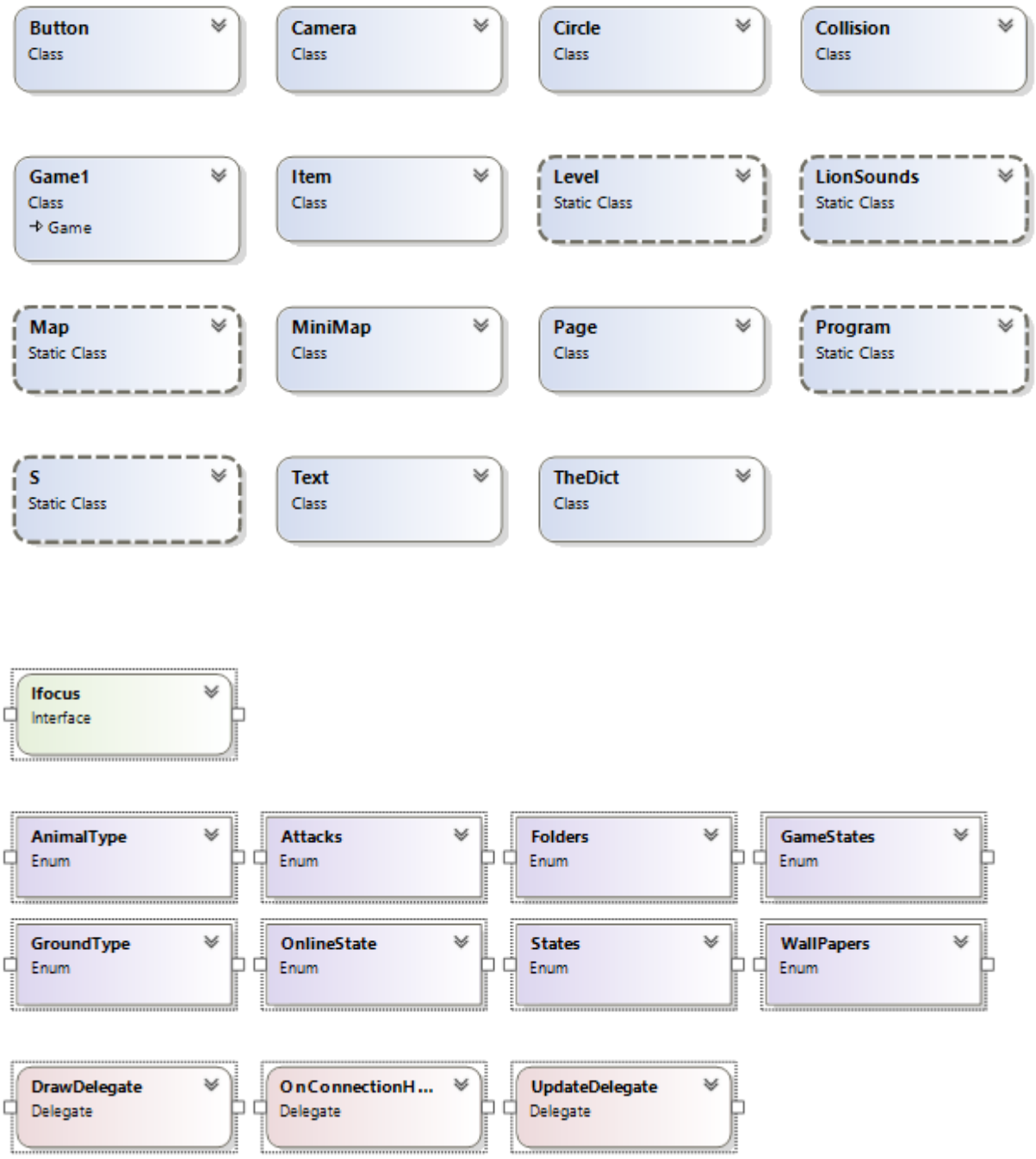
כוח הנורמל:

כוח נורמל הוא הכוח שמפעיל המשטח בתגובה לכוח שמופעל עליו.



תרשים המחלקות




Button
Class
Camera
Class
Circle
Class
Collision
Class
Game1
Class
Game
Item
Class
Level
Static Class
LionSounds
Static Class
Map
Static Class
MiniMap
Class
Page
Class
Program
Static Class
S
Static Class
Text
Class
TheDict
Class
Ifocus
Interface
AnimalType
Enum
Attacks
Enum
Folders
Enum
GameStates
Enum
GroundType
Enum
OnlineState
Enum
States
Enum
WallPapers
Enum
DrawDelegate
Delegate
OnConnectionH...
Delegate
UpdateDelegate
Delegate

- Animal.cs
- Animation.cs
- AttackBall.cs
- BaseKeys.cs
- BotKeys.cs
- Camera.cs
- Circle.cs
- Collision.cs
- Game1.cs
- HostOnlineGame.cs
- Ifocus.cs
- ImageProcess.cs
- Item.cs
- JoinOnlineGame.cs
- Level.cs
- LionSounds.cs
- Map.cs
- Menu.cs
- MiniMap.cs
- NonBaseKeys.cs
- OnlineGame.cs
- Program.cs
- SpriteObject.cs
- Static.cs
- TheDict.cs
- UserBaseKeys.cs

המחלקה הראשית – Game1

מבנה המחלקה:

תכונות –

- האובייקט graphics מסוג GraphicsDeviceManager שאחראי על התיאום על הכרטיס הגרפי וההגדרות הגרפיות של מסך המחשב.
- האובייקט spriteBatch מסוג SpriteBatch שבעזרתו מציירים טקסטורות על חלון/מסך המשחק.
- האובייקט Content מסוג ContentManager שבעזרתו טוענים את כל הקבצים של המשחק (טקסטורות, קבצי XML וכו').

פונקציות –

- **Ctor** - שתפקידו לאתחל את המשתנים.
- **LoadContent** - בפונקציה זו כותבים את כל הקוד שתפקידו לטעון את כל הקבצים.
- **Update** - שמקבל אובייקט מסוג GameTime ומתבצע 60 פעמים בשנייה ובה מבצעים את הקוד שתפקידו לעדכן ובצע פעולות שקשורות למשחק.
- **Draw** - פונקציה שמתבצעת בצורה שוטפת ומציירת את כל הטקסטורות והטקסטים למסך.

על מנת לאפשר לכל אובייקט להתנהל באמצעות עצמו ללא תלות בקריאה חיצונית מ - Game1 יצרתי 2 איוונטים.

- **UPDATE_EVENT** שבנוי לפי הDelegate – updateDelegate
- **DRAW_EVENT** שבנוי לפי הDelegate – drawDelegate

על מנת לאפשר גישה למידע בGame1 בחופשיות יצרתי מחלקה סטאטית בשם S.

המחלקה הסטאטית – S

תפקיד המחלקה הסטאטית היא לרכז את כל האובייקטים והמידע המשותף לכל האובייקטים שמשמש אותם בביצוע משימות שונות כמו ציור, זיהוי מקשים במקלדת, זיהוי לחיצה על העכבר ומיקומו.

משתנים –

- האובייקט cm מייצג את האובייקט Content הנמצא במחלקה הראשית Game1 ובכך מאפשר לכל האובייקטים לטעון בעצמם מידע.
- האובייקט spb מייצג את האובייקט spriteBatch הנמצא במחלקה הראשית Game1 האובייקטים משתמשים בו כדי לצייר טקסטורות כאשר האיוונט DRAW_EVENT מתרחש.
- האובייקט gd מייצג את הgraphicDevice בנמצא באובייקט graphics במחלקה Game1.
- המשתנה CamSpeed משתנה קבוע שמייצג את מהירות המצלמה שעוקבת אחריי הדמות שלנו.
- המשתנה MapsScale גם הוא משתנה קבוע שמייצג את ה - Scale של המפות. גם הוא קשור למצלמה ונמצא במטריצה שהיא מייצרת.

המחלקה:

```
static class S
{
        #region DATA
        public static ContentManager cm;
        public static SpriteBatch spb;
        public static GraphicsDevice gd;
        public static GraphicsDeviceManager gdm;
        public static GameStates gameState;
        public static SpriteFont GameFont;
        public static Dictionary<GameStates, Menu> AllMenues;
        public static float MapsScale;
}
```

ציור טקסטורות על המסך

המטרה: ציור טקסטורה על המסך.

היו לי שני כיוונים לממש מטרה זו:

1) לצייר בפעולה Draw שבGame1. הפעולה Draw תהיה אחראית לצייר את הטקסטורות של כל האובייקטים. ייווצר קוד מסורבל וארוך בפעולה.
2) שימוש ב - Event בשביל הציור. כל אובייקט הוא עצמי ומכיל בתוכו איוונט שקורה בכל פעם שהפעולה draw ב - Game1 מתרחשת. האובייקט הוא זה שאחראי על הציור של עצמו.

הפתרון שנבחר:

יצירת אירוע לציור טקסטורות כך שכל אובייקט יהיה אחראי לציור של עצמו. השימוש באירועים יהפוך של הקוד להרבה יותר מסודר ומאורגן ויאפשר את תחזוקתו בקלות בניגוד לרעיון הראשון שבפרויקטים גדולים יהווה גורם מפריע.

מימוש הפתרון:

הוספתי לnamespace של המשחק את הdrawDeledate - Delegate ובמחלקה Game1 יצרתי את האיוונט DRAW_EVENT.

```
public static drawDelegate DRAW_EVENT;
```

כל אובייקט שישתמש באיוונט הזה יצטרך לעשות שני דברים:

1. יצירת פעולה שכאשר האיוונט יקרה הפעולה תופעל.
2. קישור האיוונט לפעולה שלנו.

```
public void Draw ()
{
}
```

ובקונסטרקטור להוסיף את הקוד הבא:

```
Game1.DRAW_EVENT += this.Draw;
```

נניח ויש לנו טקסטורה בשם tex ואנחנו רוצים לצייר אותה בנקודה 300,300 . אז בפעולה Draw נכתוב את הקוד הבא:

```
S.spb.Draw(tex, new Vector2(300, 300), Color.White);
```

ציור הדמויות

המטרה: ציור דמויות על המסך.

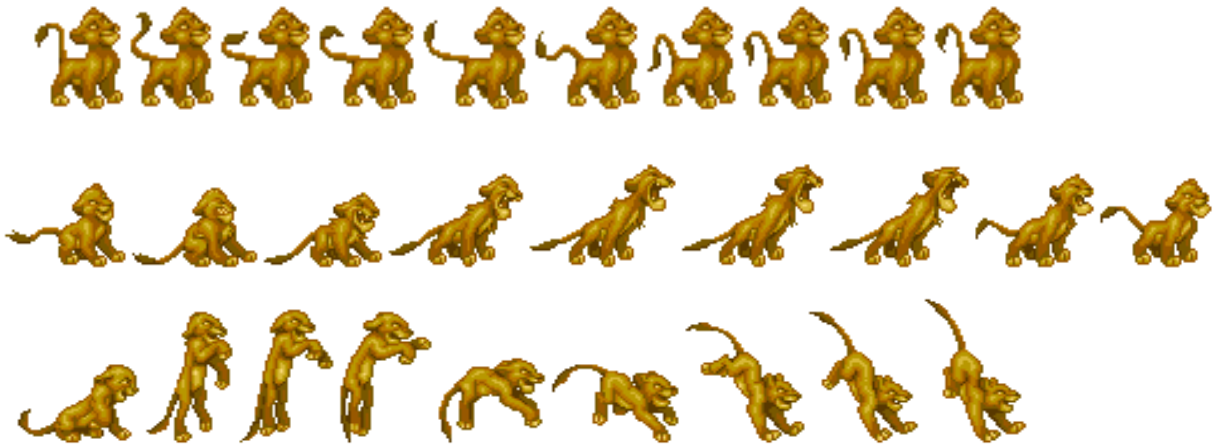
גם כאן, היו לי שני כיוונים לממש מטרה זו:

1) שימוש באוסף של טקסטורות. טעינת אוסף של טקסטורות שמרכיבות אנימציות של הדמות וכל פעם לצייר טקסטורה אחרת כדי ליצור אנימציה.
2) שימוש ב - SpriteSheets. טעינת SpriteSheet אחד לכל דמות וציור חלקים ממנו לפי Rectangle-ים ובאמצעות החלפת הRectangle-ים ליצור אנימציות.

הפתרון שנבחר:

הפתרון שנבחר הוא שימוש בSpriteSheets, הרבה יותר יעיל לטעון ולוקח פחות זמן לטעות תמונה אחת גדולה של SpriteSheet מאשר עשרות תמונות רגילות. כמו כן SpriteSheet שוקל פחות מעשרות תמונות וכך המשחק עצמו ישקול גם פחות.

מספר דוגמאות ממגוון ה - SpriteSheets של הדמות הראשית במשחק (שלב ראשון):



ניתן לראות שבכל סטריפ אנימציה יש מספר פריימים (Frames). אם הייתי בוחר בשיטת המימוש השנייה הייתי צריך לטעון מספר תמונות מסוים בהתאם למספר הפריימים שבכל סטריפ. במקרה הספציפי לעיל, לכל שלושת הסטריפים ביחד הייתי צריך לטעון 28 תמונות שונות.

יצירת שלב – מחלקת Level

מחלקה (סטאטית) זו אחראית על "יצירת העולם" על פי השלב. יש במשחק שני שלבים לכן ישנן שתי פונקציות סטאטיות אשר מאתחלות את כל המידע הדרוש לאותו שלב, הן מבחינת מפה ועיבודה, והן מבחינת מיקומים, הדמות הראשית (סימבה כשהוא גור/בוגר), ואויבים.

מבנה המחלקה:

תכונות –

LevelNumber – משתנה מטיפוס int, המציין את מספר השלב.
hero – משתנה מטיפוס Animal המציין את סימבה (הגיבור).
scar – משתנה מטיפוס Animal המציין את סקאר (האויב הראשי, מופיע בשלב השני).
lizard – משתנה מטיפוס Animal המציין זיקית.
hedgehog – משתנה מטיפוס Animal המציין את קיפוד.
beetle – משתנה מטיפוס Animal המציין את חיפושית.
Characters – רשימה מטיפוס Animal המכילה את כל האויבים, בהתאם לשלב.
BackGroundImage – משתנה מטיפוס Texture2D המציין את המפה שתוצג למשתמש.
LogicBackGround – משתנה מטיפוס Texture2D המציין את המפה הלוגית.
minimap – משתנה מטיפוס MiniMap המציין את המפה המוקטנת.
cam – משתנה מטיפוס Camera המציין את המצלמה.

פונקציות –

InitFirstLevel – מאתחלת את השלב הראשון במידע המתאים.
InitSecondLevel – מאתחלת את השלב השני במידע המתאים.

קוד המחלקה:

```
public static int LevelNumber;
  public static Animal hero;
  public static Animal scar;
  public static Animal lizard;
  public static Animal hedgehog;
  public static Animal beetle;
  public static List<Animal> Characters;

  public static Texture2D BackGroundImage;
  public static Texture2D LogicBackGround;
  public static MiniMap minimap;
  public static Camera cam;
```

מחלקת SpriteObject

מבנה המחלקה:

משתנים –

- **Pos** - משתנה מסוג Vector2 המציין את מיקום הטקסטורה הרצויה לציור.
- **Rot** - משתנה מסוג float המציין את הטיית הטקסטורה הרצויה לציור.
- **Texure** - משתנה מסוג Texure2D המאפשר לראות טקסטורה על המסך.
- **SourceRectangle** - משתנה מסוג Rec? השומר בתוכו את ארבעת המשתנים המהווים את מסגרת הטקסטורה.
- **Color** - משתנה מסוג Color המציין את צבעה של הטקסטורה המיוחלת לציור.
- **Origin** - משתנה מסוג Vector2 מציין את מרכז הטקסטורה.
- **Scale** - משתנה מסוג Vector2 המשפיעה על גודלה של הטקסטורה ע"י שתי משתנים מסוג float.
- **Effects** - משתנה מסוג Sprite Effects הגורם להיפוכה של התמונה ביחס לערכו.
- **layerDepth** - משתנה מסוג float המציין את מיקומה של הדמות בציר הz בין (0-1) ובכך בונה את התצוגה כשכבות.

פונקציות –

- **Ctor** - שתפקידו לאתחל את תשעת המשתנים הללו.
- **DrawObject** - פעולה זו אינה מקבלת דבר, אך בעזרת תכונות האובייקט וע"י הפעולה המובנת S.spb.Draw() היא מציירת את הטקסטורה על המסך.

המחלקה:

```
#region Data

 public Texture2D texture { get; set; }
 public Vector2 Pos { get; set; }
 public Rectangle? sourceRectangle;
 public Color color;
 public float Rot { get; set; }
 public Vector2 origin { get; set; }
 public Vector2 scale;
 public SpriteEffects effects;
 public float layerDepth;

 #endregion
```

מחלקת TheDict

על מנת לממש את יכולות המחלקה Animation (על מחלקה זו יבוא הסבר בהמשך) עלינו לתווך בינה לבין השחקן. כלי עזר אשר יכול לבוא לעזרתנו הוא ה - Dictionary. ה - Dictionary הוא אובייקט היוצר לשחקן מן מילון, שבכל דף קיים משתנה מסוג Animation המקנה מידע על סוג State מסוים ע"פ תמונת ה - Sprite של ה - State המסוים.

לדוגמה:

לשחקן (הדמות) במשחק שלי, קיימים מספר מצבים (States), אציג את כמה מהם, אשר הוצגו כבר לעיל:

עמידה

שאגה

קפיצה

כחלק מההסבר על אופן המימוש והביצוע, אציג קטעים מהקוד שבמחלקה זו:

```
public static Dictionary<Folders, Dictionary<States, Page>> dic;
```

כאן אני בעצם מקצה מילון של מילונים (סטאטי). כל אחד מהמילונים שבתוך המילון הגדול יהיה שייך לדמות אחרת במשחק. בתוך כל מילון תימצא תיקייה של כל אחד מן הדמויות שאליה ישתייכו כל ה – States של הדמות המסוימת. במילים אחרות, לכל דמות במשחק יש תיקייה ובתוכה כל הסטריפים של האנימציות (שאלה בעצם המצבים, ה - States) השייכות לדמות ספציפית מסוימת. כלומר, כל עמוד, כלומר כל Page (על מחלקה זו יבוא הסבר בהמשך) ב - Dictionary של כל דמות במשחק מציין את המידע הדרוש עבור State מסויים, כגון נקודות ה - origin הנקודות היוצרות את ה - Rectangle ועוד.

הסיבה שאני מחזיק מילון של מילונים היא בעצם שכל המידע לגבי כל הדמויות יהיה מאוגד במקום אחד, כדי שתהיה לי גישה ישירה אחת ומיידית לכל אחד מהמצבים של כל דמות. אם הייתי משתמש במילון אחד עבור כל דמות, היה נוצר מצב שבו יש לי מספר רב של מילונים שאין ביניהם קשר, הם היו זרים זה לזה מבלי שהיה אמצעי שהיה מאגד אותם יחד וקושר אותם, דבר שאינו יעיל ו/או נוח למתכנת.
בדרך זו תהיה לי גישה ישירה לכל מילון של כל דמות דרך מילון המילונים.

להלן קטע קוד השייך לפעולה הסטאטית Init במחלקת TheDict המציג את מה שהוסבר לעיל:

```
foreach (Folders folder in Enum.GetValues(typeof(Folders)))
            {
                Dictionary<States, Page> temp = new Dictionary<States,
Page>();

                foreach (States state in
Enum.GetValues(typeof(States)))
                {
                    try
                    {
                        temp.Add(state, new Page(folder, state));
                    }

                    catch { }
                }

                dic.Add(folder, temp);
            }
```

איך הקוד עובד:

בעבור כל תיקייה שנמצאת לי ב – Content, אני מקצה מילון זמני, אשר כל מילון זמני כזה בעצם ישמש כמילון של המצבים של כל דמות. לאחר מכן, אני עובר כל ה – States בכל תיקייה, ומכניס כל State למילון הזמני שהוקצה. בסיום סריקת כל ה – States של כל תיקייה, אני מוסיף את המילון הזמני (אשר בתוכו כבר יש מידע בעבור דמות מסוימת) למילון הכללי, למילון המילונים.

* חשוב שהשם של כל State יהיה אחד השמות המופיע במאגר ה – States הכללי, המוגדר כ – enum, מפני שמתבצעת קריאה של ערכי enum, הנקראים כמחרוזת, וערכים אלה מושווים לערכים של השמות של כל State.

```
    enum States
    {
        Stand, Running, Running_Jump, Roar, Jump, Falling, AfterFalling,
        Hanging,   Climb,   Slash,   Pouncing,   Rolling,   DoubleSlash,
GettingHurt,   GettingSlashed,   Crouch,   CrouchSlash,   ThrowingEnemy,
TossedBySimba, BeetleBeforeDying
    }
```

מחלקת Page (ImageProcess)

מחלקת Page (או מחלקת "עיבוד תמונה"), היא בעצם העמוד בתוך כל Dictionary, בתוך כל מילון שבמילון המילונים הראשי הכולל. בכל עמוד יש את המידע הרלוונטי הדרוש של כל State ההכרחי לעיבוד התמונה ולבסוף ציורה על המסך, כפי שהוסבר לעיל בעמודים הקודמים.

מבנה המחלקה:

משתנים –

- **tex** - משתנה מסוג Texture2D המציין את תמונות הsprites המחולק לפי מצבים (ריצה, עמידה, מעידה וקפיצה).
- **rec** - משתנה מסוג רשימה של Rectangle, כל איבר ברשימה מציין את הנקודות ע"פ הtex שיוצרות Sprite מסויים.
- **Org** - משתנה מסוג רשימה של Vector2, כל איבר מציין את מיקום מרכזו של הsprite המסוים.
- **FlippedOrg** - משתנה מסוג רשימה של Vector2, כל איבר מציין את מיקום מרכזו של הSprite המסוים במקרה של שינוי ב - SpriteEffects.
- **BigCircles** - משתנה מטיפוס circle (על מחלקה זו יבוא הסבר בהמשך).
- **SmallCircles** - משתנה מטיפוס circle.
- **AllSmallCircles** - משתנה מטיפוס circle.
- **IsFound** - משתנה מסוג bool.
- **name** - משתנה מסוג string, המציין את השם של הטקסטורה.

פונקציות –

- **Ctor** - שמקבל שם Folder ושם של State (הנמצא בתוך ה - Folder) ותפקידו לאתחל את המשתנים, לבצע Processing (עיבוד) לטקסטורה ולבצע לה שיקוף.
- **DefineStatesSlows** – קביעת המהירות שבה תתבצע החלפות בין כל ה - Frame של כל State.
- **Processing** – ביצוע עיבוד לטקסטורה.
- **MakeTrans** – ביצוע שיקוף לטקסטורה.

המחלקה:

```
public List<Rectangle> rec { get; private set; }
public List<Vector2> org { get; private set; }
public List<Vector2> FlippedOrg { get; private set; }
public Texture2D tex { get; private set; }
public List<Circle> BigCircles;
public List<Circle> SmallCircles;
public List<List<Circle>> AllSmallCircles;
public bool IsFound;
public string name;
```

עיבוד טקסטורה:

כאן נכנס ההסבר הוויזואלי של יצירת האנימציה.

ניקח לדוגמה את סטריפ האנימציה של ה – State ← שאגה.

הנקודות השחורות המוצגות בתמונה הן מצדי כל Frame ובנקודת הייחוס, ה – origin.
בצורה זו אני תוחם כל Frame בכל Sprite ומכניס אותו ל - List של Rectangle-ים בהתאם לכל State. כמו כן, אני מכניס לרשימה את כל הנקודות השחורות ככה שבהמשך על ידי חישוב וקוד מתאים אכניס את מיקומי נקודות הייחוס, ה – Origins לרשימה. ניתן לשים לב שברוב ה – Frames, כל נקודה שחורה המציינת סוף של Frame מסוים, מציינת גם את תחילתו של ה – Frame הבא. ובכך בעצם מתבצעת תחימה של Frame ב – Rectangle, אשר מוכנסים לרשימה אחת.

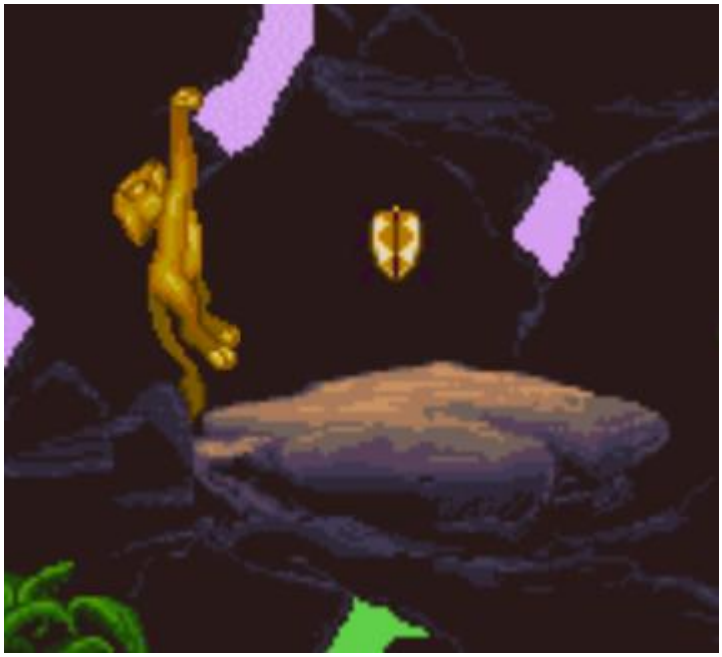
ישנו מקרה יוצא דופן אחד שעליו ארצה לפרט והינו מצבים שבהם ה – States הם של התלייה והטיפוס.

כפי שמוצג בתמונה משמאל, זהו ה – State של התלייה.
על פי הדרך שהוצגה לעיל, אכן יש נקודות שחורות בצדי הטקסטורה ובנקודת ה – Origin. אך ניתן לראות שישנה עוד נקודה שחורה, באזור כף ידו של סימבה, באותו טור שבו נמצאת נקודת ה – Origin.
הסיבה לכך שישנם States (כגון זה), שבהם ארצה שנקודת הייחוס תהיה שונה, על מנת שהטקסטורה תוצג בצורה מסוימת במשחק.
לכן ארצה שלרשימת נקודות ה – Origins, תיכנס אותה נקודה שנמצאת באזור כף ידו של סימבה, ולא הנקודה השחורה באמצע שנמצאת בתחתית הטקסטורה.

תלייה

אם לא הייתי פועל בדרך זו במקרים המסוימים האלה, מצב התלייה על פלטפורמה במשחק היה נראה במקום כך (המצב הרצוי והנכון):

היה נראה כך (המצב השגוי):



ניתן לראות שסיטואציה כזאת פוגעת באיכות ובחוויית המשחק.

להלן קטע הקוד המטפל בכל המוצג לעיל:

```
public void Processing(string name)
        {
            tex = S.cm.Load<Texture2D>(name);
            Color[] col = new Color[tex.Width];
            Color[] check = new Color[tex.Width * tex.Height];
            tex.GetData<Color>(0, new Rectangle(0, tex.Height - 1,
tex.Width, 1), col, 0, tex.Width);
            tex.GetData<Color>(check);

            // מערך של כל הנקודות השחורות בסטריפ מסוים
            List<int> pnt = new List<int>();

            for (int i = 0; i < col.Length; i++)
            {
                if (col[i] == col[0])
                {
                    // מוסיף את מיקומי הנקודות האלה כדי שאוכל להשתמש בהם בהמשך
                    pnt.Add(i);
                }
            }

            IsFound = false;

            // עובר כל ה - Origins
            for (int i = 1; i < pnt.Count; i += 2)
            {
                for (int row = 0; row < tex.Height - 2; row++)
                {
                    if (check[pnt[i] + (row * tex.Width)] == col[0])
                    {
                        IsFound = true;
                        org.Add(new Vector2(pnt[i] - pnt[i - 1],
row));
                        FlippedOrg.Add(new Vector2(pnt[i + 1] -
pnt[i], row));

                        break;
                    }
                }

                if (!IsFound)
                {
```

```
                    org.Add(new Vector2(pnt[i] - pnt[i - 1],
tex.Height - 1));
                    FlippedOrg.Add(new Vector2(pnt[i + 1] - pnt[i],
tex.Height - 1));
                }

                rec.Add(new Rectangle(pnt[i - 1], 0, pnt[i + 1] -
pnt[i - 1], tex.Height - 2));
            }
        }
```

<u>איך הקוד עובד:</u>

מתבצעת טעינה של הטקסטורה של ה – State המסוים.
מוקצים שני מערכי מטיפוס Color. לאחד המערכים מוכנס המידע לגבי כל הצבעים שנמצאים בשורה התחתונה של סטריפ האנימציה, ה – State המסוים, ולמערך השני כל המידע לגבי הצבעים שבכל הטקסטורה המסוימת. כל הנקודות השחורות מוכנסות לרשימה (pnt) מטיפוס int.
עובר כל נקודות ה - Origins ובודק בנוסף אם קיימת נקודה שחורה אחרת, שלא שייכת לשורה התחתונה של הטקסטורה, על מנת שנוכל להתייחס למקרים יוצאי הדופן שפורטו לעיל (תלייה וטיפוס), בנקודות ייחוס שונות, כלומר נקודות Origin שונות. אם מצאנו נקודה כזאת, החישובים המבוצעים והמידע שנכנס יהיו בהתאם אליה, ואם לא מצאנו, אז המידע מוכנס בהתאם לנקודת ה – Origin הנמצאת בכל מקרה בשורה התחתונה בכל טקסטורה (ברירת המחדל).

*רשימת ה – Rectangle-ים של כל State מסוים תישאר זהה, בן אם מצאנו נקודת ייחוס שונה ובין אם לא, מפני שגם במקרה שנמצאת נקודת ייחוס אחרת, אין לדבר זה כל השפעה על ה – Rectangle-ים התוחמים כל Frame.

ביצוע שיקוף לטקסטורה:

פעולת makeTrans פעולה זו מקבלת תמונה ו"מעלימה" בה את כל גווני הצבע הלא רלוונטיים לטקסטורה.

פעולה זו לוקחת את הפיקסל הראשון (הפיקסל בצד השמאלי העליון) של הטקסטורה, סורקת את כל הפיקסלים האחרים, מבצעת השוואה בין הפיקסל הראשונה לפיקסל שהתקבל, ובמקרה של שוויון היא משנה אותו לצבע שקוף, כלומר - Color.Transparent.

דבר זה "מנקה" את הפיקסלים של רקע התמונה ומשאירים אותה "נקייה" בעלת הפיקסלים הרלוונטיים בלבד.

קטע הקוד:

```
public void makeTrans()
  {
      Color[] data = new Color[tex.Width * tex.Height];
      tex.GetData<Color>(data);
      Color trans = data[0];
      Color black = Color.Black;
      for (int i = 0; i < data.Length; i++)
      {
          if (data[i] == trans)
          {
              data[i] = Color.Transparent;
          }

          // התנאי הזה הוא בשביל שלא יראו את הנקודות השחורות על גוף הדמות
         // של השחקן בזמן טיפוס על פלטפורמה
          if (data[i] == black && i < data.Length - 1)
          {
              data[i] = data[i - 1];
          }
      }

      tex.SetData<Color>(data);
  }
```

לדוגמה:

קביעת מהירויות החלפת Frame-ים ב – State מסוים:

כפי שהוסבר קודם, על ביצוע מטרה זו, אחראית הפונקציה הסטטית DefineStatesSlows.

פעולה זו מקבלת State מסוים, וקובעת באיזו מהירות הפריימים ב – State הזה יוחלפו. המהירות היא ביחידות של X/60 שנייה.

```
public static int DefineStatesSlows(States state)
{
     if (state == States.Running)
     {
         return 3;
     }

     if (state == States.Running_Jump)
     {
         return 6;
     }

     .
     .
     .
     States - המשך קביעת מהירויות לכל שאר ה
     .
     .
     .

}
```

כלומר, אם ה – State הוא של State הריצה, אז המהירות שבה תתבצע החלפת ה – Frames היא 3/60 שנייה. כלומר בכל 1/20 שנייה, ישנה קפיצה לפריים הבא, ולאחר מכן לבא שאחריו וכו' (וכך בצורה מעגלית ,אם יש צורך בכך, לדוגמה ב – State של העמידה, כלומר כשמגיעים לפריים האחרון אז הבא אחריו הוא הראשון).

מחלקת Animation

הנפשה (או החייאה) היא תחום העוסק בהקניית אשליה של תנועה על גבי מסך.

שימוש נרחב בהנפשות נעשה בקולנוע, בטלוויזיה ובאתרי אינטרנט.

תפקידה של ההנפשה היא ל"רמות" את עיני הרואה במספר תמונות (Sprites) המשתנות ברגע ובכך ליצור אשליה של תנועה.

במשחק שלי, יש צורך בדבר זה במגוון מצבים, כגון:

- ריצה
- קפיצה
- עמידה
- שאגה
- גלגול
- ועוד...

מחלקת Animation היא זו שיוצרת את האפשרות לביצוע אנימציה במשחק.

מבנה המחלקה:

משתנים –

- **folder** - משתנה מסוג Folder המציין את תיקיית ה – States של דמות מסוימת.
- **state** – משתנה מסוג State המציין את ה – State המסוים.
- **PrevState** – משתנה מסוג State המציין את ה – State הקודם של דמות מסוימת.
- **IsAnimationover** – משתנה מסוג bool המציין אם האנימציה של State מסוים הסתיימה.
- **index** - משתנה מסוג Int, משתנה המציין את מיקומו של ה – Frame הספציפי ב – State מסוים.
- **indexSlow** – משתנה מסוג int אשר בעצם משמש כמונה, כצובר הנועד לבדיקת שוויון עם המשתנה slow, על מנת לדעת באיזו מהירות (מהירות החלפת ה – Frames לשנייה) אנו נמצאים כרגע.
- **slow** - משתנה מסוג int, משתנה המציין את מהירות השינוי שלה - Frames בשנייה.

פונקציות –

- **Ctor** - מקבל 11 משתנים. חלק מן המשתנים מאותחלים כתוצאה מקשרי הורשה בין המחלקות Animation ו – SpriteObject. המשתנים המפורטים לעיל, חוץ מ – PrevState, מאותחלים בתוך ה – Ctor (כלומר לא מעצם קשרי ההורשה).
- **DrawObject** - פונקציה זו כלומר אינה מקבלת פרמטרים. בקוד של פונקציה זו ניתן לראות שמימוש הפונקציה יהיה אחר, היות ואני דורס את המימוש של פונקציה זו במחלקה שממנה ירשתי (SpriteObject), על ידי שימוש ב – override. פעולה זו אחראית על ציור האובייקט / הטקסטורה על פי כל המידע הנחוץ שקיים לגביה.

המחלקה:

```
public Folders folder { get; set; }
public States state { get; set; }
public States PrevState;
public bool IsAnimationOver;
public int index;
int indexSlow;
int slow;
```

מחלקת Map

מחלקת "מפה" אחראית בעצם על עיבוד מידע אודות מפה מסוימת.
הדרך שבה בחרתי ליצור את המשחק במובנים של: מתי יהיה אפשר ללכת / לקפוץ (פלטפורמה), מתי אי אפשר להתקדם מעבר למקום מסוים כגון – מכשול / סלע שחוסם את הדרך), מתי יהיה אפשר להיתלות וכדומה – היא בשיטת בדיקת צבע מסוים.

כלומר, במפות הלוגיות שלי, ישנם אזורים צבועים בצבעים מסוימים אשר כל צבע מסמל אזור הנועד למטרה מכוונת אחת. למשל, האזורים הצבועים בצהוב, הם בעצם הפלטפורמות, המקומות שבהם ניתן ללכת, לרוץ, לקפוץ עליהם. האזורים האדומים הם אזורים שלא ניתן לחצות / לעבור אותם, הם בעצם מהווים את המחסומים / המכשולים. האזורים הכחולים (אשר מאוד קשה לראותם ללא זום, מפני שהם בעצם פיקסל אחד שצבוע) הם האזורים שבהם דמות הגיבור יכולה להיתלות, וכו'.

מבנה המחלקה:

משתנים –

- **col** - משתנה מסוג Color המכיל את כל צבעי הפיקסלים בכל המקומות שבמפה מסוימת.
- **Locations** – מטריצה מסוג *GroundType המכיל את מיקומי כל הנקודות/המיקומים שיש בטקסטורה (מפה) מסוימת.
- **HangLocations** – רשימה מסוג Vector2 המכיל את כל מקומות האפשריים בהם אפשר להיתלות.
- **SwingLocations** – רשימה מסוג Vector2 המכיל את כל מקומות האפשריים בהם ניתן להיתלות תוך כדי התנדנדות.
- **tex** - משתנה מסוג Texture2D, המכיל את הטקסטורה, המפה.

* GroundType – זהו enum (המוגדר ברמת ה – namespace) שמכיל את כל מצבי השטח האפשריים במשחק שלי.

```
enum GroundType {Platform, Air, Blocked, Can_Hang, Swing}
```

פונקציות –

check – פונקציה סטאטית המקבלת את טקסטורת המפה ומעבדת את המידע לגביה.

המחלקה:

```
static Color[] col;
public static GroundType[,] Locations;
public static List<Vector2> HangLocations;
public static List<Vector2> SwingLocations;
public static Texture2D tex { get; private set; }
```

אופן עיבוד המידע:

את המטריצה Locations אני מאתחל בגודל של מספר הפיקסלים באורך הטקסטורה על רוחב הטקסטורה. דבר דומה מתבצע למערך החד ממדי col, שאליו מוכנסים כל צבעי הפיקסלים שכמותם היא מכפלת מספר הפיקסלים באורך הטקסטורה ברוחב הטקסטורה. את רשימות הווקטורים של מיקומי התלייה והתנדנדות אני מאתחל גם בהתאם.

בלולאת for מקוננת העוברת על אורך ורוחב הטקסטורה אני בודק לגבי כל פיקסל את צבעו. כאן באופן ישיר מתחרש עיבוד המידע. אני בעצם משווה את צבעו של כל פיקסל לאחד מארבעת הפיקסלים הראשונים הנמצאים בצד השמאלי העליון של הטקסטורה.

ב – col[0] נמצא הצבע – צהוב.
ב – col[1] נמצא הצבע – אדום.
ב – col[2] נמצא הצבע – כחול.
ב – col[3] נמצא הצבע – ירוק (קיים רק במפה השנייה).

ואחרי כל השוואה לכל אחד מארבעת הפיקסלים הללו, אני מכניס את המידע הדרוש למטריצת ה – enum, הלא היא – Locations.
לגבי הפיקסלים המציינים את האזורים שבהם אפשר להיתלות או להתנדנד, בנוסף להכנסת מידע זה למטריצת ה – enum, אני מכניס מיקומים אלה לרשימות המיקומים (הווקטורים) המתאימות. אם אף אחד מצבעי הפיקסלים בטקסטורה לא שווה לאחד מארבעת הפיקסלים לעיל, אני כברירת מחדל מכניס פיקסלים אלה ומציין ששטחים המכילים אותם הם "אוויר" (Air).

<u>להלן קוד הפונקציה המטפל בעניין זה:</u>

```
public static void check(Texture2D tex)
{
    Locations = new GroundType[tex.Height, tex.Width];
    HangLocations = new List<Vector2>();
    SwingLocations = new List<Vector2>();
    col = new Color[tex.Width * tex.Height];
    tex.GetData<Color>(col);

    for (int i = 0; i < tex.Height; i++)
    {
        for (int j = 0; j < tex.Width; j++)
        {
            if (col[tex.Width * i + j] == col[0])
            {
                Locations[i, j] = GroundType.Platform;
            }
            else if (col[tex.Width * i + j] == col[1])
```

```
                {
                    Locations[i, j] = GroundType.Blocked;
                }
                else if (col[tex.Width * i + j] == col[2])
                {
                    HangLocations.Add(new Vector2(j, i));
                    Locations[i, j] = GroundType.Can_Hang;
                }
                else if (col[tex.Width * i + j] == col[3])
                {
                    SwingLocations.Add(new Vector2(j, i));
                    Locations[i, j] = GroundType.Swing;
                }
                else
                {
                    Locations[i, j] = GroundType.Air;
                }
            }
        }
    }
```

מחלקת MiniMap

מחלקה זו אחראית על ציור המפה הקטנה בצד השמאלי העליון של מסך המשחק.
יש צורך ב- mini map מכיוון שהרי היות והמפה המוצגת למשתמש, המפה ש"עליה" הוא משחק, נתונה לו בהגדלה, ב – zoom מסוים, או כשם המשתנה במשחק שלי – Scale. כלומר, המשתמש לא רואה את מפת המשחק בכללותה ברגע נתון, אלא רק חלק / מקטע מסוים שלה.
למפות במשחק יש Scale מסוים אשר מציג אותן בגודל מוגדל מכפי שהן במקור, וזאת לצורך חוויית המשחק. המפה המוקטנת, מציגה בעצם באופן מלא את כל אזורי המפה המקורית, דבר המעניק נוחות למשתמש להבין כיצד הוא נמצא בכל רגע נתון, ולראות איזה חלק יחסי מהמפה הוא הספיק לעבור.

מבנה המחלקה:

משתנים –

zoom – משתנה מטיפוס float המציין את הזום שבו תוקטן המפה המקורית. התמונה המוקטנת היא בעצם ה – Minimap.

spbo – משתנה מטיפוס SpriteObject (מופע של המחלקה האחראית, כידוע, על ציור טקסטורה).

color – משתנה מטיפוס Color, שיציין את צבע ה – Rectangle המסמל דמות מסוימת, על המפה המוקטנת.

פונקציות –

Ctor – שמקבל 2 משתנים, tex ו – scale. תפקידו לאתחל את מופע המחלקה spbo שיכיל בעצם מידע על המפה המוקטנת, בערכים המתאימים.

Draw – פונקציה אשר מציירת בפועל את ה – mini map בצד השמאלי העליון של מסגרת המשחק.

כמו כן, ארצה לציין פונקציה אשר אמנם אינה מוגדרת במחלקה זו, אך מופיעה בפונקציה Draw.
הפונקציה המדוברת הינה draw_rect והיא זאת שמציירת את ה – Rectangle-ים המציינים את הדמויות, על גבי המפה המוקטנת (פונקציה זו מופיעה במחלקה SpriteObject).

אם הדמות הינה דמות הגיבור במשחק, צבע ה – Rectangle שלו יצויר בצבע אדום על ה – mini map, אחרת (כלומר כל האויבים), צבעו יהיה כחול.

להלן הקוד המתאים:

```
if (Game1.Characters[i].AnimalType == AnimalType.Adult_Simba ||
    Game1.Characters[i].AnimalType == AnimalType.Cub_Simba)
{
    color = Color.Red;
}
else
{
    color = Color.Blue;
}
```

קטע הקוד המצייר את ה – Rectangle-ים המוסברים לעיל:

```
SpriteObject.draw_rect(new Vector2(Game1.Characters[i].Pos.X * zoom,
                    Game1.Characters[i].Pos.Y * zoom - 2), 8, color);
```

להמחשה, ארצה להציג את מפת השלב הראשון כשהיא בתמונת המקור, לבין אותה המפה אשר נתונה ב – Scale מסוים.

בגודלה המקורי:

ב – Scale נתון:

* למפות נתתי Scale של 2.5f , כלומר מפות המשחק מוגדלות בזום של פי 2.5 מגודלן המקורי.

תמונה להמחשת ה – mini map במשחק:

במשחק שלי, מצב זה מתאפשר בלחיצה (ממושכת) על מקש Tab.

מחלקת Camera

המטרה היא שהמצלמה תעקוב אחרי השחקן בתנועתו במישור האופקי והאנכי, במישור ה - XY.

ישנן 2 אפשרויות למימוש מטרה זו.
הראשונה, היא ליצור תנועה מזויפת של השחקן. הדמות של השחקן תמיד ימצא בנקודה מסוימת על המסך ומקום להזיז אותו נזיז את המפה ואת שאר האובייקטים לכיוון הפוך מכיוון התנועה שלו כך שתיווצר אשליה של תנועה של השחקן.
השנייה, היא שימוש במטריצות.

הפתרון שנבחר הוא הפתרון השני, שימוש במטריצות כדי ליצור מצלמה אמיתית שתעקוב אחריי השחקן.

מבנה המחלקה:

משתנים –

- **Mat** - משתנה מטיפוס מטריצה (Matrix).
- **Focus** – משנתה מטיפוס Ifocus המציין על מיקומו וזוויתו של הדמות שאחריה המצלמה עוקבת.
- **Zoom** - משתנה מטיפוס float המציין על זום המצלמה.
- **Pos** - משתנה מטיפוס Vector2 המציין על מיקום המצלמה.
- **View** - מטיפוס Viewport המציין על גודלה של המצלמה ומיקומה על חלון המשחק.

פונקציות –

- **Ctor** - שתפקידו לאתחל את המשתנים על ידי ה - Ifocus שמקבל הזום וה - View. בנוסף מוסיף את ה - update event ל - events.
- **Update** - פעולה זאת מעדכנת את תכונות המצלמה בהתאם לתפקוד המשחק.

המחלקה:

```
class Camera
 {
     public Matrix Mat { get; private set; }
     public Ifocus Focus { get; private set; }
     public Vector2 Zoom { get; private set; }
     public Viewport View { get; private set; }
     public Vector2 Pos { get; private set; }

}
```

מימוש הפתרון:

```
namespace xxx
{
    class Camera
    {
        #region Data
        public Matrix Mat { get; private set; }
        public Ifocus Focus { get; private set; }
        public Vector2 Zoom { get; private set; }
        public Viewport View { get; private set; }
        public Vector2 Pos { get; private set; }
        #endregion

        /// <summary>
        /// Initializing the parameters
        /// </summary>
        /// <param name="focus"></param>
        /// <param name="zoom"></param>
        /// <param name="view"></param>
        public Camera(Ifocus focus, Vector2 zoom, Viewport view)
        {
            Game1.UPDATE_EVENT += this.UpdateMat;

            Focus = focus;
            Zoom = zoom;
            View = view;
            Pos = Focus.Pos;
        }

        /// <summary>
        /// Updating the cam's data
        /// </summary>
        public void UpdateMat()
        {
            Mat = Matrix.CreateTranslation(-Pos.X, -Pos.Y, 0) *
                  Matrix.CreateScale(Zoom.X, Zoom.Y, 1) *
                  Matrix.CreateTranslation(View.Width / 2, View.Height / 2, 0);

            if (Level.LevelNumber == 1)
            {
                Pos = Vector2.Lerp(new Vector2(
                    MathHelper.Clamp(Focus.Pos.X, (80 + View.Width / 2), 3350),
                    MathHelper.Clamp(Focus.Pos.Y, -300 + View.Height, 3170 - View.Height
/ 2)), Pos, 0.8f);
            }

            if (Level.LevelNumber == 2)
            {
                Pos = Vector2.Lerp(new Vector2(
                    MathHelper.Clamp(Focus.Pos.X, (View.Width / 2), 3360),
                    MathHelper.Clamp(Focus.Pos.Y, -2000 + View.Height, 4000 - View.Height
/ 2)), Pos, 0.8f);
            }
        }
    }
}}
}
```

איך הקוד עובד:

הקונסטרקטור – מקבל את האובייקט שאחריו המצלמה תעקוב שיורש את הinterface, את ה - Viewport של המסך שעליו מציירים את כל האובייקטים ואת המיקום ההתחלתי של המצלמה ומכניס אותם למשתנים - Focus, View, Position.

הפעולה UpdateMat - מתרחשת כל פעם שה - update ב - Game1 מתבצע והיא מעדכנת את המטריצה של המצלמה. המטריצה של המצלמה מורכבת ממכפלה של 3 מטריצות משני סוגים: Matrix.CreateTranslation, MatrixCreateScale , כאשר:

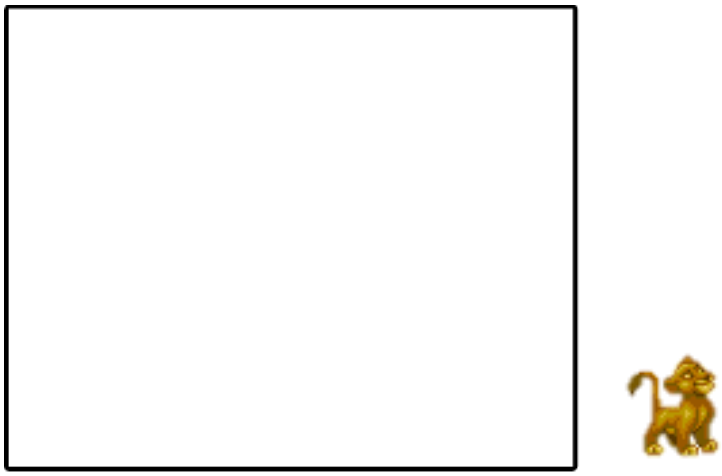
Matrix.CreateTranslation – משמש לתזוזה.

Matrix.CreateScale – משמש כדי לשנות את הגודל של מה שאנחנו מציירים.

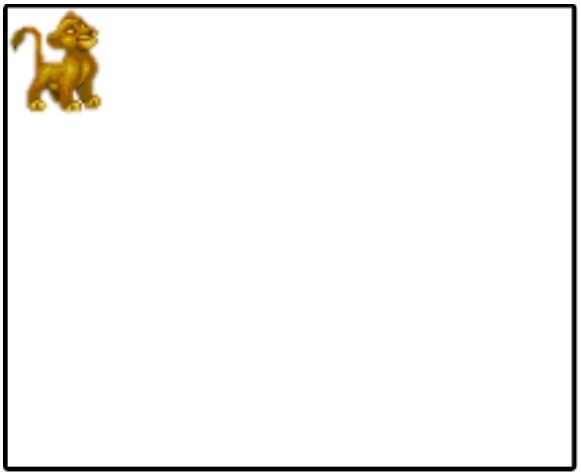
איך המצלמה פועלת?

תחילה לוקחים את מה שמציירים כמו שהוא. מכיוון שאנחנו רוצים שהדמות תמיד תהיה במרכז אנחנו עושים CreateTranslation וכך מעבירים את כל הציור ככה שהדמות נמצאת בתחילת המסך בנקודה ה שמאלית העליונה – (0,0). מפעילים את הScale הרצוי באמצעות createScale ואז מזיזים את כל הציור למרכז המסך עם עוד createTranslation.

כלומר, נניח שהריבוע השחור הוא המסך והדמות שלנו נמצא מחוצה לו:

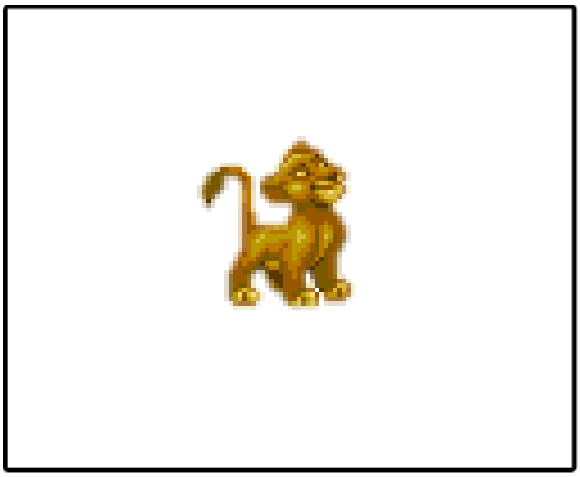


עכשיו אנחנו רוצים לבצע CreateTranslation כך שכל מה שאנחנו מציירים יזוז, והדמות שלנו תהיה ב - (0,0), כלומר ← `Matrix.CreateTranslation(-Pos.X, -Pos.Y, 0)` .



לאחר מכן אנחנו רוצים להפעיל את ה- Scale על כל מה שאנחנו מציירים – `Matrix.CreateScale(Zoom.X, Zoom.Y, 1)` .

ובסופו של דבר המטרה הסופית למקם את כל הדברים שאנו מציירים כך שהדמות שלנו נמצאת באמצע.
`Matrix.CreateTranslation(View.Width / 2, View.Height / 2, 0)`.



מחלקת Circle

התנגשות במשחק

התנגשות בין דמויות:

אנו רוצים לגרום לדמות להתנגש בעצמים אשר נמצאים במשחק. לשם כך נגדיר מחלקה שתעזור לנו במימוש מטרה זו.

מבנה המחלקה:

משתנים –

Center – משתנה ווקטורי שמסמן את מרכז המעגל שאנו רוצים ליצור.

Radius – משתנה מטיפוס float המציין את רדיוס המעגל.

פונקציות –

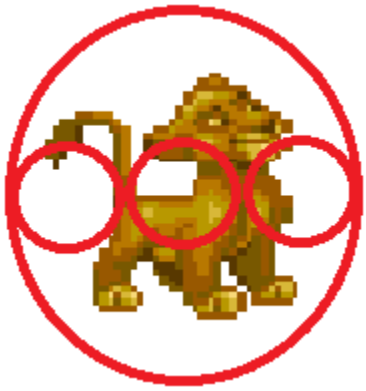
Ctor – מקבל את האובייקט מרכז המעגל, Rectangle שמציין Frame מסוים (ה – Rectangle שבו תחומה טקסטורה מסוימת), scale, ומשתנה בוליאני המציין האם זה עיגול גדול. הפונקציה מאתחלת את משתני המחלקה על ידי אובייקטים אלה ומזמנת את הפונקציה find_radius לחישוב הרדיוס.

find_radius – הפונקציה מקבלת שלושה פרמטרים (texture, scale, Boolean isBig), ומחשבת את גודל הרדיוס.

במידה וזה עיגול גדול גודל הרדיוס יהיה החלק הגדול מבין רוחב התמונה לגובה התמונה .

במידה וזה עיגול קטן הרדיוס יהיה החלק הקטן מבין הרוחב והגובה.

find_end, **find_shloshtreve**, **find_reva**, **find_center** – פונקציות סטטיות שמוצאות נקודה על תמונה (רבע, אמצע {חצי}, שלושה רבעים).



לדוגמה, דמות הגיבור ומסביבו עיגול גדול שבתוכו שלושה עיגולים קטנים.

מחלקת Collision

לאחר שהגדרנו לכל אובייקט במשחק עיגולים גדולים וקטנים. נוכל כעת לבדוק האם יש התנגשות בין עיגולים אלה ובכך להגדיר האם האובייקטים מתנגשים.

מבנה המחלקה:

פונקציות –

CheckCollision – במצב כזה המעגלים עולים אחד על השני ויש התנגשות.

מכיוון שבמשחק שלי יש אנימציות, לא מספיק התנאי הפשוט של בדיקה אם סכום הרדיוסים של שני המעגלים קטן מהמרחק בין המרכזים של הדמויות. אנו צריכים לבדוק התנגשות עם כל אחד מן העיגולים מרשימת העיגולים הראשונה לשנייה (של הגיבור והאויב) בעבור כל State של כל דמות.

כלומר, בתחילה אני בודק אם יש חשד להתנגשות, על ידי כך שאני בודק אם סכום שני הרדיוסים של העיגולים הגדולים קטן מהמרכזים של הדמויות ב – State מסוים שבה נמצא הן הגיבור והן האויב.

להלן:

```
if ((TheDict.dic[hero.folder][hero.state].BigCircles[StatesIndex1 %
TheDict.dic[hero.folder][hero.state].rec.Count].radius +
TheDict.dic[enemy.folder][enemy.state].BigCircles[StatesIndex2 %
TheDict.dic[enemy.folder][enemy.state].rec.Count].radius) >
(hero.Pos - enemy.Pos).Length())
{
.........
}
```

במידה ונכנסנו לתנאי, אני בודק התנגשות כל אחד מן העיגולים מרשימת העיגולים הראשונה לשנייה (של הגיבור והאויב).

<u>להלן:</u>

```
foreach (Circle cir1 in
TheDict.dic[hero.folder][hero.state].AllSmallCircles[StatesIndex1 %
TheDict.dic[hero.folder][hero.state].rec.Count])
{
     foreach (Circle cir2 in
TheDict.dic[enemy.folder][enemy.state].AllSmallCircles[StatesIndex2 %
TheDict.dic[enemy.folder][enemy.state].rec.Count])
     {

     }
}
```

<u>לדוגמה, התנגשות עם זיקית:</u>

```
if (enemy.AnimalType == AnimalType.lizard)
{
    if (hero.AnimalType == AnimalType.Cub_Simba)
    {
        if ((hero.jumping || hero.falling) && enemy.Pos.Y - hero.Pos.Y
<= 2 * cir2.radius && hero.Pos.X >= enemy.Pos.X - cir2.radius - 10f &&
   hero.Pos.X <= enemy.Pos.X + cir2.radius + 10f)
        {

        }
    }
}
```

* כאן ספציפית נתתי סטייה של 10f במרחק מכיוון שרציתי שההתנגשות יותר טובה.

התנגשות בין דמות למתקפה:

התנגשות בין דמות מסוימת, לבין מתקפה, לא נעשית דווקא עם עיגולים.
הסתפקתי בבדיקת מרחקים בין המתקפה לבין הדמות.

להלן:

```
if (hero.fireball != null)
{
    if (((hero.fireball.Pos - enemy.Pos).Length()) <= 90f)
    {
        enemy.state = States.GettingSlashed;

        if (hero.effects == SpriteEffects.None &&
            enemy.effects == SpriteEffects.None)
        {
            enemy.effects = SpriteEffects.FlipHorizontally;
        }

         else if (hero.effects == SpriteEffects.FlipHorizontally &&
                  enemy.effects == SpriteEffects.FlipHorizontally)
         {
              enemy.effects = SpriteEffects.None;
         }

         Game1.UPDATE_EVENT -= hero.fireball.Update;
         Game1.DRAW_EVENT -= hero.fireball.Draw;
         hero.fireball = null;
      }
  }
```

תפריטים במשחק

כפתורים וטקסטים:

מפאת נוחות כלפי המשתמש, יש להכין תפריטים שיעזרו לנווט במשחק ולתת למשתמש לקבל החלטות בנוגע לאפשרויות מסוימות שירצה המשתמש לבחור, בצורה נוחה.

קודם כל יש להגדיר , שתי מחלקות בסיסיות שעליהן יתבסס התפריט:

מחלקת Tex:

מטרת מחלקה זו היא לצייר טקסט על המסך שלא יהיה לחיץ אלא כטקסט לקריאה בלבד.

משתנים –

- **Text** – הטקסט שיהיה רשום על המסך.
- **Scale** – גודל הכיתוב.
- **Color** – צבע הכיתוב
- **Position** – מיקום הכיתוב.

פונקציות –

Ctor – מאתחל את משתני המחלקה.

Draw – ציור טקסט בסיסי עפ"י תכונות המחלקה.

מחלקת Button:

מטרת המחלקה היא לצייר טקסט על המסך שיהיה לחיץ בעזרת המקלדת.

משתנים –

- **Text** – הטקסט שיהיה רשום על המסך.
- **Name** – שם הכפתור (נחוץ בשביל לנוע בין תפריטים).
- **Position** – מיקום הכפתור.
- **Color** – צבע הכפתור.

פעולות:

Ctor – אתחול משתני המחלקה.
ReadButtonName – הפעולה שאחראית לשינוי מצבי המשחק.
drawButton – ציור בסיסי של הכפתור על פי פרמטרים.

חלק ראשון:

```
try
{
   S.gameState =(GameStates)Enum.Parse(typeof(GameStates), name);

   if (name == "MainMenu")
   {
       S.ReloadGame();
   }

 }
```

בשלב זה אנו מנסים להעביר את מצב המשחק לשם הכפתור שנלחץ לדוגמא , אם בחרתי ללחוץ על כפתור אונליין , המשחק יעבור למצב אונליין.

}

בודקים את סוג הכפתור שנלחץ ומעדכנים בהתאם.

התפריט:

לאחר הגדרת כפתורים וטקסטים למשחק , אפשר להתחיל לעצב את התפריט. לשם יצירת תפריטי המשחק , אגדיר מחלקת תפריט בסיסית שכל תפריט במשחק ירש ממנה את תכונותיה ופעולותיה.

מחלקת Menu:

משתנים –

Buttons – רשימת כל הכפתורים של התפריט.

Texts – רשימת כל הטקסטים של התפריט.

Select – האינדקס של הכפתור שכרגע מסומן בתפריט.

NumberButtons – מס' הכפתורים בתפריט.

curState - מצב המקלדת הנוכחי.

lastState – מצב המקלדת הקודם.

פונקציות –

Update – מעדכן את הבחירה כשנלחצים מקשי החיצים. בנוסף , כשנלחץ המקש Enter הפעולה קוראת לפעולה שקוראת את שם הכפתור ומעבירה תפריט.

draw – ציור של כל הרשימה buttons.

כעת, לאחר שמוגדרות מחלקות הכפתורים, וגם הוגדרה מחלקת תפריט בסיסי, בכל מחלקת תפריט יש צורך להגדיר רשימת כפתורים ורשימת טקסטים והמחלקה הבסיסית כבר תדע לבצע את הפעולות הנדרשות.

מחלקת MainMenu:

מחלקה זו יורשת את מחלקת menu ומקבלת את כל תכונותיה ופעולותיה.

פונקציות –

Ctor – הפעולה מאתחלת את הפרמטרים של המחלקה הבסיסית. מוסיפה כפתורים לרשימת buttons וטקסטים לרשימת texts.

ובעזרת הפעולות של המחלקה הבסיסית נוצר התפריט הבא:

מחלקת BaseKeys

לשם כך שהדמות תזוז , יש להורות לה לעשות זאת. הוראה זו ניתנת על ידי לחיצה על מקשים (במקרה של בן אדם זוהי המקלדת הפיזית ובמקרה של מחשב זוהי מקלדת וירטואלית).

מחלקה זו היא מחלקה אבסטרקטית שיהיו שני סוגי שחקנים שיממשו אותה: מחשב ואדם.

מחלקה אבסטרקטית הינה מחלקה אשר מהווה בסיס למחלקות אחרות, אך אין לנו באמת צורך באובייקטים ממנה בתוכנית.

כמה נקודות לגבי מהותה של מחלקה אבסטרקטית:

- בכדי ליצור מחלקה אבסטרקטית יש לכתוב **abstract** לפני שם ה – class.
- לא ניתן ליצור אובייקטים ממחלקה אבסטרקטית.
- ניתן ליצור ייחוס ממחלקה אבסטרקטית (בד"כ לצורכי פולימורפיזם).
- מחלקה אבסטרקטית יכולה להכיל כל דבר שמחלקה רגילה יכולה להכיל.
- מחלקה אבסטרקטית יכולה (אבל לא חייבת) להכיל פונקציות אבסטרקטיות.
- לא ניתן לכתוב פונקציה אבסטרקטית במחלקה שהיא לא אבסטרקטית.
- פונקציה אבסטרקטית היא פונקציה שאין לה מימוש. היא נכתבת כהצהרה ללא גוף (כלומר ללא בלוק של פקודות).
- כאשר יורשים מחלקה אבסטרקטית חובה לממש את כל הפונקציות האבסטרקטית שבה (אם יש).
- המימוש של פונקציה אבסטרקטית הוא באמצעות מילת המפתח **override**.

פונקציות –

8 פעולות אבסטרקטיות שיממשו את לחיצות השחקן:

UpKey – האם השחקן קפץ.
DownKey – האם השחקן התכופף.
LeftKey – האם השחקן ביצע תזוזה שמאלה.
RightKey – האם השחקן ביצע תזוזה ימינה.
ShiftKey – האם השחקן לחץ על מקש Shift (השמאלי). מקש זה הוא אחד משני מקשים (קומבו) היוצרים מתקפה מסוימת.
PunchtKey – האם השחקן תוקף (מבצע התקפה).
FireBalltKey – האם השחקן ביצע מתקפת כדור אש.
FireBalltKey – האם השחקן ביצע מתקפת כדור קרח.

קוד המחלקה:

```
abstract class BaseKeys
{
    public abstract bool UpKey();
    public abstract bool DownKey();
    public abstract bool LeftKey();
    public abstract bool RightKey();
    public abstract bool ShiftKey();
    public abstract bool PunchKey();
    public abstract bool FireBallKey();
    public abstract bool IceBallKey();
}
```

מחלקת UserBaseKeys

מחלקה זו היא מימוש של המחלקה BaseKeys ע"י שחקן שהוא בן אדם.

משתנים –

UpKey – משתנה Keys שמייצג האם השחקן קפץ.
DownKey – משתנה Keys שמייצג האם השחקן התכופף.
LeftKey – משתנה Keys שמייצג האם השחקן ביצע תזוזה שמאלה.
RightKey – משתנה Keys שמייצג האם השחקן ביצע תזוזה ימינה.
ShiftKey – משתנה Keys שמייצג האם השחקן לחץ על מקש Shift (השמאלי). מקש זה הוא אחד משני מקשים (קומבו) היוצרים מתקפה מסוימת.
PunchtKey – משתנה Keys שמייצג האם השחקן תוקף (מבצע התקפה).
FireBalltKey – משתנה Keys שמייצג האם השחקן ביצע מתקפת כדור אש.
FireBalltKey – משתנה Keys שמייצג האם השחקן ביצע מתקפת כדור קרח.

פונקציות –

Ctor – מאתחל את כל המקשים על ידי פרמטרים.

כמה דוגמאות על סמך אותו עיקרון למימוש כל הפעולות של BaseKeys על ידי בדיקה אם המקש לחוץ:

```
public override bool UpKey()
{
    return Keyboard.GetState().IsKeyDown(Up);
}

public override bool DownKey()
{
    return Keyboard.GetState().IsKeyDown(Down);
}

public override bool LeftKey()
{
    return Keyboard.GetState().IsKeyDown(Left);
}

public override bool RightKey()
{
    return Keyboard.GetState().IsKeyDown(Right);
}
```

מחלקת Animal

מחלקה זו יורשת את מחלקת Animation ומקבלת את כל תכונותיה ופעולותיה.

מחלקה זו אחראית על תפעול חיה מסוימת, דמות מסוימת במשחק.
כלומר מתבצעת כאן בדיקה של כל המצבים והאפשרויות של שביכולתה של דמות מסוימת לעשות, כגון – ריצה, קפיצה, תקיפה וכדומה.

מבנה המחלקה:

תכונות –

AnimalType - משתנה מטיפוס AnimalType enum המציין את סוג החיה.
NumOfSeconds - משתנה מטיפוס float המציין את מספר השניות שהדמות באוויר, בהנחה שיש לה יכולת קפיצה.
gravity - משתנה מטיפוס const float המציין את קבוע התאוצה.
RightRunSpeed - משתנה מטיפוס float המייצג את המהירות הריצה בכיוון הימני שבה תתקדם דמות מסוימת.
LeftRunSpeed - משתנה מטיפוס float המייצג את המהירות הריצה בכיוון השמאלי שבה תתקדם דמות מסוימת.
jumpspeed - משתנה מטיפוס float המייצג את המהירות שבה הדמות תבצע את פעולת הקפיצה.
jumping - משתנה מטיפוס bool המציין את הדמות כרגע אכן בקפיצה.
falling - משתנה מטיפוס bool המציין את הדמות כרגע אכן בנפילה (לאחר תלייה).
Generalfalling - משתנה מטיפוס bool המציין את הדמות כרגע אכן בנפילה (כללית).
Hanging - משתנה מטיפוס bool המציין את הדמות כרגע אכן תלויה על פלטפורמה.
Swinging - משתנה מטיפוס bool המציין את הדמות כרגע אכן תלויה בהתנדנדות על פלטפורמה.
PressedFire - משתנה מטיפוס bool המציין אם אחד ממקשי היריות הוקש.
PressedPunch - משתנה מטיפוס bool המציין אם מקש ה – punch הוקש.
PressedDirectionToClimb - משתנה מטיפוס bool המציין אם אחד ממקשי הכיוונים (חץ שמאלי/ימני) הוקש בזמן שהדמות תלויה, על מנת לטפס.
LeftRuning - משתנה מטיפוס bool המציין אם הדמות רצה לכיוון שמאל.
RightRuning - משתנה מטיפוס bool המציין אם הדמות רצה לכיוון ימין.
LeftStanding - משתנה מטיפוס bool המציין אם הדמות נעמדת לכיוון שמאל.
RightStanding - משתנה מטיפוס bool המציין אם הדמות נעמדת לכיוון ימין.
WasJumping - משתנה מטיפוס bool המציין אם הדמות הייתה לאחרונה בקפיצה.
WasFalling - משתנה מטיפוס bool המציין אם הדמות הייתה לאחרונה בנפילה.
Crouching - משתנה מטיפוס bool המציין אם הדמות נמצאת במצב התכופפות.
RunningJump - משתנה מטיפוס bool המציין אם הדמות נמצאת במצב קפיצת ריצה.
Rolling - משתנה מטיפוס bool המציין אם הדמות נמצאת במצב התגלגלות.
DoubleSlashing - משתנה מטיפוס bool המציין אם הדמות נמצאת במצב מכה כפולה.
ThrowingAway - משתנה מטיפוס bool המציין אם הדמות נמצאת במצב הטלה.
DelayBetweenFires - משתנה מטיפוס float המציין את העיכוב שיש לחכות בין לחיצה ללחיצה כאשר רוצים לירות.
DelayBetweenSlashes - משתנה מטיפוס float המציין את העיכוב שיש לחכות בין לחיצה ללחיצה כאשר רוצים לתקוף (מכה).

baseKeys - משתנה מטיפוס BaseKeys המציין מקלדת המקשים הרלוונטית לכל דמות על מנת לבצע פעולות שונות נחוצות במשחק.
fireball - משתנה מטיפוס AttackBall המייצג את כדור האש.
iceball - משתנה מטיפוס AttackBall המייצג את כדור הקרח.
effect - משתנה מטיפוס SpriteEffects המייצג את האפקט של כדור המתקפה המסוים.
SimbaEffectBeforeThrowing - משתנה מטיפוס SpriteEffects המייצג את האפקט של סימבה (הגיבור) לפני שמבצע הטלה (במצב בוגר).
IndexOfCreatureThatWasThrown - משתנה מטיפוס int המציין את המיקום של החיה שהוטלה במערך האויבים הסטאטי.
StartFalling_Y - משתנה מטיפוס float המציין את המיקום בציר ה-ה Y של דמות מסוימת לפני נפילתה.
EndFalling_Y - משתנה מטיפוס float המציין את המיקום בציר ה-ה Y של דמות מסוימת לאחר נפילתה.

* לא כל התכונות שלעיל רלוונטיות לכל אחת מן הדמויות, אלא רק לחלק מן החיות/הדמויות. כל אחת בהתאם למה שביכולתה לבצע.
** הפעולות שדמות מסוימת מבצעת, נבדקות באמצעות States. לדוגמה, אם באחד ההסברים מטה יהיה רשום (לדוגמה) שפעולה מסוימת לא יכולה להתבצע בזמן פעולת "ריצה", הכוונה היא שאותה הפעולה לא מתבצעת כאשר ה – State שונה מ – States.Running .

<u>פונקציות –</u>

Update – פונקציה האחראית לעדכון כל הקשור והנחוץ לדמות/חיה מסוימת.
ShootFireBall – פונקציה שמטרתה לייצר מתקפת כדור אש.
ShootIceBall – פונקציה שמטרתה לייצר מתקפת כדור קרח.
DrawAnimal – פונקציה האחראית על ציור הדמות/החיה.
Fall – אחראית על ביצוע הנפילה, בהתאם לכללי הפיזיקה.

תהליכים מרכזיים אצל דמות:

- תזוזה:

תזוזה מתאפשרת רק אם אחד ממקשי התזוזה לחוצים והדמות לא מבצעת שום פעולה אחרת (למעט קפיצה והתכופפות) וגם אם אין **מול** הדמות אזור שיחסום אותה מלהמשיך. כלומר אם הדמות נמצאת בכיוון ריצה מסוים, ובעוד מרחק של כמה פיקסלים (היסט/ווקטור) לא קיים שטח/אזור ובו פיקסלים אדומים, ניתן להמשיך לנוע לכיוון זה (כפי שהוזכר לעיל, בנושא - מחלקת Map).

להלן קטע הקוד הרלוונטי:

```
if ((baseKeys.RightKey() || baseKeys.LeftKey()) &&
!baseKeys.FireBallKey() && !baseKeys.IceBallKey() &&
!Hanging && !Swinging && !Rolling && !Crouching && this.state !=
States.GettingHurt && this.state != States.AfterFalling &&
this.state != States.Slash && this.state != States.DoubleSlash &&
this.state != States.CrouchSlash &&
this.state != States.ThrowingEnemy && this.state != States.Roar &&
this.state != States.Pouncing &&
((this.effects == SpriteEffects.None &&
Map.Locations[(int)(Pos.Y / S.MapsScale), (int)((Pos.X + 15f) /
S.MapsScale)] != GroundType.Blocked) ||
(this.effects == SpriteEffects.FlipHorizontally &&
Map.Locations[(int)(Pos.Y / S.MapsScale), (int)((Pos.X - 15f) /
S.MapsScale)] != GroundType.Blocked)))
{

. . .

}
```

* היסט הפיקסלים המסוים הינו 15 פיקסלים, המוספים / מוחסרים ממיקום הדמות בציר ה – X, בהתאם לכיוון תנועתו.

** כפי שהוסבר לעיל, מתאפשרת תזוזה רק אם הדמות לא מבצעת שום פעולה אחרת, למעט קפיצה והתכופפות, מכיוון ששילוב המצבים יוצר מצב חדש.

לדוגמה, משילוב תזוזה לכיוון מסוים + קפיצה, נוצר המצב – "קפיצת ריצה".
להלן ההבדלים בין שני סוגי הקפיצות:

משילוב תזוזה לכיוון מסוים + התכופפות, נוצר המצב – "גלגול".
להלן:

התכופפות

גלגול

- קפיצה:

תהליך הקפיצה הינו תהליך מאוד משמעותי כאשר בזמן ביצוע פעולת קפיצה, מתרחשות מגוון בדיקות תוך כדי התהליך. תהליך הקפיצה יכול להתרחש, בדומה לתזוזה, כשהדמות לא מבצעת כמעט אף פעולה למעט כמה פעולות בודדות, כגון יכולת קפיצה בזמן גלגול (כלומר סיטואציה כזו – מתאפשרת).

להלן הקוד המתאר כיצד הקפיצה פועלת:

```
if (jumping)
{
    Pos += new Vector2(0, jumpspeed);

    if (jumpspeed <= 25f) // הגבלת מהירות
    {
        jumpspeed += 0.6f; // באיזה קצב תהיה הקפיצה
    }
    . . .
}
```

בזמן הקפיצה, נעשית בדיקה אם תלייה תהיה אפשרית, כלומר נבדוק אם כרגע אנחנו לא במצב תלייה כבר, ואם יש אזור עם פיקסל בודד כחול במרחק מסוים מן הדמות וכל זאת תוך התייחסות לכיוון של הדמות ביחס לנקודת התלייה (SpriteEffect) – וכל זאת תוך כדי שאנחנו עוברים בלולאה על כל נקודה אפשרית כזו ברשימת הנקודות המיוחסות לאזורי התלייה האפשריים. ברגע שמצאנו, אנו יוצאים מהלולאה מיד.

להלן:

```
foreach (Vector2 hangingData in Map.HangLocations)
{
    if (!Hanging && jumpspeed >= 0 &&
       Math.Abs((new Vector2(Pos.X / S.MapsScale, Pos.Y / S.MapsScale)
      - new Vector2(hangingData.X, hangingData.Y)).Length()) <= 15f &&
      ((this.effects == SpriteEffects.FlipHorizontally &&
      Map.Locations[(int)(hangingData.Y),
      (int)((hangingData.X - 5f))] == GroundType.Platform) ||
      (this.effects == SpriteEffects.None &&
      Map.Locations[(int)(hangingData.Y),
      (int)((hangingData.X + 5f))] == GroundType.Platform)))
     {
         Hanging = true;
         jumping = false;
         this.state = States.Hanging;
         this.Pos = new Vector2(hangingData.X * S.MapsScale,
                                hangingData.Y * S.MapsScale);

         break;
      }
}
```

בדיקה דומה נעשית לגבי הנקודות ברשימת הנקודות המיוחסות לאזורי ההתנדנדות האפשריים.

* במשחק שלי, רק סימבה הגור וסימבה הבוגר מסוגלים להיתלות, ורק סימבה הבוגר יכול להתנדנד.

בנוסף, מתבצעת בדיקת נפילה מגובה. אם המרחק בציר ה – Y של הנפילה הוא מרחק כולל של יותר מ – 200 פיקסלים, אז ה – State של סימבה ייהפך ל – State של "לאחר נפילה" (AfterFalling).

להלן:

הבדיקה נעשית על ידי זה שרגע לפני הבדיקה, אני שומר את הערך של הדמות בציר ה – Y, ולאחר סיום נפילה / קפיצה, אני בודק את ערך ה – Y שוב, ואם ההפרש עולה על 200f, אז this.State = States.AfterFalling .

בסיום כל אנימציה מסוימת, כברירת מחדל הדמות נעמדת (נמצאת ב – State של עמידה).

* State זה ("לאחר נפילה") אפשרי רק לגבי סימבה (גור ובוגר).

בנוסף, כל זמן שדמות מסוימת נמצאת באוויר, נעשית בדיקה של האם ניתן להמשיך ללכת על פלטפורמה, במידה והיא משופעת, במידה מסוימת.

```
for (int k = 0; k < 10; k++)
{
    if (Map.Locations[(int)((Pos.Y - k) / S.MapsScale),
       (int)(Pos.X / S.MapsScale)] == GroundType.Platform && !jumping)
    {
        this.Pos = new Vector2(Pos.X, Pos.Y - k);
    }
}
```

<u>הסבר:</u>

בכל נקודה שבה נמצא השחקן, נעשית בדיקה של - האם במרחק של עד 10 פיקסלים מעל נקודת ה – Origin של הגיבור (כלומר 10 פיקסלים במורד ציר ה - Y), יש פיקסלים צהובים (שטח המוגדר כפלטפורמה), אם כן – הגיבור יעמוד בנקודה הגבוהה ממנו המקסימלית, בטווח הזה.

יצרתי סיטואציה כזו, כדי לפתור קושי שנתקלתי בו – כאשר הדמות הגיעה לפלטפורמה משופעת, והמשיך לנוע לכיוונה – הדמות נפלה, מכיוון שהדמות מוגדרת לזוז ישר בציר ה – X, ללא התייחסות לציר ה – Y, בעת תזוזתה. דבר זה פגם באיכות ובחוויית המשחק.

<u>המחשה:</u>

לדוגמה, בסיטואציה כזו, סימבה היה נופל למטה (מפני שכברירת מחדל כשדמות נמצאת באוויר, היא מוגדרת כ – falling), כי הוא לא מזהה "עלייה", גם אם בסדר גודל קטן.

** במורד פלטפורמה משופעת הכל היה תקין, מפני שמיד כשהדמות נופלת, היא פוגשת בפלטפורמה מתחתיה, וכך נוצרת האשליה של ירידה במורד פלטפורמה משופעת.

לאחר הקוד המטפל בבעיה זו, נוצר המצב התקין:

בינה מלאכותית

בינה מלאכותית היא ענף של מדעי המחשב העוסק ביכולתם של מחשבים לפעול באופן המציג יכולות השמורות עד כה לבינה האנושית בלבד. במשחק שלי, קיימת בינה מלאכותית המדמה מחשבה אנושית ושיקול דעת אנושי וכן קבלת החלטות בהתאם לסיטואציה מסוימת.

במשחק, חיות בסדר גודל של סימבה (צבועים או סקאר) יש יכולת חשיבה מובנת ומורכבת יותר, לעומת החיות הקטנות (קיפודים, חיפושיות, לטאות).
החיות הגדולות מסוגלות לעקוב אחרי סימבה ויש להן את היכולת למצוא את הדרך העדיפה והכדאית להשיג את סימבה.

ארצה לפרט על האלגוריתם שכתבתי המפרט את התהליך המרכזי של מציאת בעבור הדרך העדיפה ביותר להגעה אל סימבה, בעבור אויב (בסדר גודל של סימבה).

ניקח מצב לדוגמה את סקאר, הרודף אחרי סימבה (הגור). שיקול קבלת החלטותיו הינו כדלקמן:

דבר ראשון, סקאר האויב מקבל נקודת מבט ראשונית לגבי מיקומו של סימבה, כלומר, בודק אם הוא משמאלו או מימינו, על מנת לדעת מה הכיוון ההתחלתי אליו סקאר צריך להתקדם.
לאחר מכן, נעשית בדיקה מי משניהם נמצא במקום יותר גבוה מהשני (מקום יותר גבוה, משמעותו שיעור – Y יותר נמוך, מפני שמערכת הצירים בסביבת XNA מוגדרת כך שהכיוון החיובי של ציר ה – Y הינו כלפי מטה, לעומת מנועי משחק וסביבות משחק אחרות).

במידה והאויב מעל סימבה, הוא יחפש בקרבתו מקום לרדת או ליפול לעבר המיקום של סימבה, בהנחה שסימבה נמצא בקרבת מיקום שכזה בהפרש של 100 פיקסלים לשני הצדדים ממנו והלאה. אם לא מצא בקרבת מקום אזור שכזה, סקאר יישאר במיקום יחסי של ציר ה – Y של סימבה, ויבין שישתלם לחכות לסימבה למעלה, כי בסופו של דבר סימבה יצטרך לעשות את דרכו כלפי מעלה מתישהו.

במידה והאויב מתחת לסימבה, הוא יחפש דרכים אפשריות לעלות ולהתקדם לעבר סימבה. בדרכו, נתקל האויב בתנאי שטח המצריכים אותו להפעיל שיקול דעת של מה כדאי לו לעשות. במידה והאויב מתקרב לאזור המוגדר כ – "אוויר", האויב יקפוץ, בתקווה שיצליח להגיע לפלטפורמה המחכה לו בקצה.
במידה ונתקל המוגדר כ – "חוסם" (אזור עם פיקסלים אדומים), יש שני אופציות.
הראשונה, האויב בודק אם יש ביכולתו לקפוץ מעל אזור זה ולבדוק אם יוכל להשיג את הפלטפורמה שמחכה לו בהמשך מיד אחרי האזור החוסם.
השנייה, במידה ואין ביכולתו לקפוץ מעל אזור זה (כלומר נעשית בדיקה של מה נמצא כאשר האויב מגיע לגובה הקפיצה המקסימלי), הוא מבין שהדרך היחידה להתקדם היא רק על ידי פנייה והתקדמות לעבר הכיוון הנגדי לכיוון שבו היה עד היתקלותו באזור החוסם.

*כמובן שבכל תזוזה שסימבה מבצע, שיקול החלטותיו של האויב מתעדכן בהתאם, כ – 60 פעם בשנייה.
** ההיסט (offset) שמוסף / מוחסר ממיקומו של האויב בציר ה - X בעת בדיקת התקרבות לאזורי אוויר או אזורים חוסמים, הוא 15 פיקסלים, וזאת על מנת שקצת לפני שהוא מגיע לקצה הפלטפורמה – שיבצע קפיצה.
*** ההיסט (offset) שמוסף / מוחסר ממיקומו בציר ה - Y של האויב בעת בדיקת יכולת קפיצה מעל אוזר חוסם בגובה מסוים, הוא 220 פיקסלים, כלומר גובה הקפיצה המקסימלי.

לגבי חיות קטנות לעומת סדר גודלו של סימבה, שיקול דעתן הוא להלן:

הן מתקדמות הלוך וחזור כל מרחק מסוים, ללא תלות במיקום של סימבה.
לטאות מסוגלות לתקוף ברצף ללא הפסקה כאשר סימבה נמצא בטווח הראייה שלהן.

להלן קטע הקוד האחראי על תנועתן של חיות אלה:

```
public void Walk()
{
    if (this.animal.Pos.X < Start_X)
    {
        TurnRight = true;
        TurnLeft = false;
    }

    if (this.animal.Pos.X > Start_X + 550f)
    {
        TurnLeft = true;
        TurnRight = false;
    }

    if (TurnRight)
    {
        right = true;
        left = false;
    }

    if (TurnLeft)
    {
        left = true;
        right = false;
    }
}
```

רגע לפני שסימבה נפגע מתקיפת הלטאה

רגע אחרי שסימבה נפגע מתקיפת הלטאה

להלן קטע הקוד השייך לבינת החיות בסדר גודלו של סימבה:

```
if (FirstCheck)
{
    if (this.animal.Pos.X > this.Simba.Pos.X)
    {
       left = true;
       WasLeft = true;
       WasRight = false;
       right = false;
    }

    else
    {
        right = true;
        WasRight = true;
        WasLeft = false;
        left = false;
    }

    FirstCheck = false;
}

if (this.animal.Pos.Y < this.Simba.Pos.Y)
{
      SimbaIsUnderEnemy = true;
      SimbaIsAboveEnemy = false;
}


else if (this.animal.Pos.Y > this.Simba.Pos.Y)
{
      SimbaIsAboveEnemy = true;
      SimbaIsUnderEnemy = false;
}

else
{
      SimbaAndEnemyIsInTheSameHight = true;
      SimbaIsAboveEnemy = false;
      SimbaIsUnderEnemy = false;
}
```

```
if (SimbaAndEnemyIsInTheSameHight)
{
      if (this.animal.Pos.X < this.Simba.Pos.X && this.Simba.Pos.X -
      this.animal.Pos.X >= 100f && this.Simba.Pos.Y ==
      this.animal.Pos.Y)
      {
            right = true;
            WasRight = true;
            left = false;
            WasLeft = false;
      }

      else if (this.animal.Pos.X > this.Simba.Pos.X &&
      this.animal.Pos.X -              this.Simba.Pos.X >= 100f &&
      this.Simba.Pos.Y == this.animal.Pos.Y)
      {
            left = true;
            WasLeft = true;
            right = false;
            WasRight = false;
      }

      else
      {
            stop = true;
            right = false;
            left = false;
            up = false;
      }
}

if (Math.Abs(this.Simba.Pos.Y - this.animal.Pos.Y) <= 5f &&
   ((this.Simba.Pos.X > this.animal.Pos.X && this.Simba.Pos.X -
   this.animal.Pos.X <= 100f) ||
   (this.Simba.Pos.X < this.animal.Pos.X && this.animal.Pos.X -
   this.Simba.Pos.X <= 100f)))
{
    stop = true;
}
```

```
if (SimbaIsUnderEnemy || SimbaIsAboveEnemy)
                {
    stop = false;

    if (WasRight && !stop)
    {
        right = true;
        left = false;
        WasLeft = false;
    }

    else if (WasLeft && !stop)
    {
       left = true;
       right = false;
       WasRight = false;
    }

    if (SimbaIsAboveEnemy)
    {
        if ((Map.Locations[(int)(this.animal.Pos.Y / S.MapsScale),
                (int)((this.animal.Pos.X - 15f) / S.MapsScale)] ==
           GroundType.Blocked && Map.Locations[(int)((this.animal.Pos.Y
           - 220f) / S.MapsScale),
               (int)((this.animal.Pos.X - 15f) / S.MapsScale)] !=
           GroundType.Blocked) ||
           (Map.Locations[(int)(this.animal.Pos.Y / S.MapsScale),
               (int)((this.animal.Pos.X + 15f) / S.MapsScale)] ==
           GroundType.Blocked && Map.Locations[(int)((this.animal.Pos.Y
           - 220f) / S.MapsScale),
               (int)((this.animal.Pos.X + 15f) / S.MapsScale)] !=
           GroundType.Blocked))
        {
            up = true;
        }
        else if (Map.Locations[(int)(this.animal.Pos.Y / S.MapsScale),
                (int)((this.animal.Pos.X - 15f) / S.MapsScale)] ==
GroundType.Air || Map.Locations[(int)(this.animal.Pos.Y/ S.MapsScale),
                (int)((this.animal.Pos.X + 15f) / S.MapsScale)] ==
                GroundType.Air)
        {
              up = true;
        }
```

```
        else
        {
              up = false;
        }
    }

    if ((Map.Locations[(int)(this.animal.Pos.Y / S.MapsScale),
              (int)((this.animal.Pos.X + 15f) / S.MapsScale)] ==
      GroundType.Blocked && Map.Locations[(int)((this.animal.Pos.Y -
      220f) / S.MapsScale), (int)((this.animal.Pos.X + 15f) /
      S.MapsScale)] == GroundType.Blocked))
    {
         left = true;
         right = false;
         WasLeft = true;
         WasRight = false;
    }

    if ((Map.Locations[(int)(this.animal.Pos.Y / S.MapsScale),
              (int)((this.animal.Pos.X - 15f) / S.MapsScale)] ==
      GroundType.Blocked && Map.Locations[(int)((this.animal.Pos.Y -
      220f) / S.MapsScale), (int)((this.animal.Pos.X - 15f) /
      S.MapsScale)] == GroundType.Blocked))
    {
        right = true;
        left = false;
        WasRight = true;
        WasLeft = false;
    }

    if (SimbaIsUnderEnemy)
    {
        if (this.Simba.Pos.X > this.animal.Pos.X && this.Simba.Pos.X -
this.animal.Pos.X >= 100f)
        {
            right = true;
            left = false;
            WasRight = true;
            WasLeft = false;
        }
```

```
            if (this.Simba.Pos.X < this.animal.Pos.X &&
            this.animal.Pos.X - this.Simba.Pos.X >= 100f)
            {
                left = true;
                right = false;
                WasLeft = true;
                WasRight = false;
             }
        }
    }
}
```

אונליין

במשחק קיימת אפשרות לשחק ברשת , כל שחקן ישחק יכול להשתתף ממחשבו האישי ולהתחבר למשחק. המשחק יהיה בדיוק אותו הדבר כמו משחק מקומי (אוף ליין).

האונליין פועל בעזרת פרוטוקול TCP server , שבו נשלח ונקרא מידע מן הסרבר על ידי השחקנים.

ישנה מחלקה OnlineGame שהיא מכילה את המידע הבסיסי עבור שימוש באונליין.

ישנן 2 מחלקות שממשות את מחלקה זו , מחלקת host , join. אחד הוא יוצר המשחק והשני מצטרף אל המשחק.

המחלקה OnlineGame:

משתנים –

- **Reader** – ערוץ שדרכו אפשר לקלוט נתונים.
- **Writer** – ערוץ שדרכו אפשר לכתוב נתונים.
- **Thread** – תהליך, שמטרתו לגרום לתעבורת נתונים ברשת.
- **Client -** השרת עצמו במשחק.
- **Port** – הפורט שבו השרת פועל.
- **hostCar, JoinCar** – האריה (השחקן) במשחק.
- **OnConnection** – זה Event שפועל כששחקן מצטרף למשחק.

פונקציות –

- **Init** – מאתחלת את מכוניות המשחק ואת הthread.
- **InitChars** – פונקציה אבסטרקטית שמטרתה היא אתחול השחקנים במשחק, כל משתמש יאתחל את האריה שלו.
- **RaiseOnConnectionEvent** – מפעילה Event שמפעיל את המשחק.
- **SocketThread** – פעולה אבסטרקטית שמטרתה היא להתחיל את המשחק, ולתפעל את כתיבת הנתונים.
- **StartCommunication** – הפעולה מאתחלת את ה - thread בצורה הבאה:

```
public void StartCommunication()
{
    thread = new (newThreadStart(SocketThread));
    thread.IsBackground = true;
    thread.Start();
}
```

זוהי פונקציה בעלת משמעות חשובה. ארצה להרחיב לגבי תפקידו ה – thread. זהו בעצם תהליך שמוגדר להתבצע בכל חייו. בשיטה זו התהליך ירוץ ברקע, כלומר בו זמנית כשהמשחק רץ. בכך אנו שולחים וקוראים נתונים בלי לפגוע בפעילות ובחוויית המשחק ובצורה נוחה.

במהלך הפעולה אנו מריצים את ה - thread ובכך מזמנים את הפונקציה socketThread.

עכשיו בנינו את כל התשתית לביצוע משחק רשתי ונשארו שתי הפעולות המרכזיות שגורמות למידע לעבור ברשת: קריאת נתונים ושליחת נתונים:

- **ReadAndUpdateChars** – פונקציה זו מקבלת דמות שאנו רוצים לקרוא אליה נתונים מהשרת, ומעדכנת את הנתונים שלה בעזרת הreader stream, ומבצעים קריאה בהתאם לנתונים שנשלחו.
- **WriteCharachterData** – כמו פונקציית ה - read, פונקציה זו מקבלת דמות של שחקן שרוצה לכתוב את נתוניו אל הסרבר. הכתיבה נעשית באמצעות writer stream כלומר נתיב שפתוח לכתיבה בשרת.

יצירת משחק אונליין

לאחר שהגדרנו משחק אונליין בסיסי, עלינו לממש את הפעולות האבסטרקטיות במחלקת הבסיס (onlineGame) אשר מורישה למחלקה זו את תכונותיה.

המחלקה HostOnlineGame:

פונקציות –

- **InitChars** – הפונקציה מאתחלת את השחקנים במשחק, ואת מקשיהן. בפועל נוצרות שני אריות, אחת לשימוש ע"י השחקן היוצר (host) , בעלת מקשים רגילים של המקלדת הפיזית, ואחת של השחקן המשחק איתנו ברשת. דמות זו היא בלי יכולת תזוזה ממשית במשחק שלנו, ולשם כך יצרתי את המחלקה - **NonBaseKeys** - מחלקה שמטרתה לא לחיצות על המקלדת בעבור שחקן מסוים, לבחירתי. כלומר יוצרת מקלדת וירטואלית ללא מקשים.

דוגמה:

```
hostChar.baseKeys = new UserBaseKeys(Keys.Up, Keys.Down, Keys.Left,
Keys.Right, Keys.LeftShift, Keys.Space, Keys.LeftControl,
Keys.LeftAlt);

joinChar.baseKeys = new NonBaseKeys();
```

- **SocketThread** – במצב של יצירת המשחק, תפקידה לאתחל את הסרבר על ידי פתיחתו.

להלן:

```
TcpListener listener = new TcpListener(IPAddress.Any, port);
listener.Start();
client = listener.AcceptTcpClient();
```

בהתחלה מבוצע כאן אתחול של tcp והפעולה מחכה לקבל משתמש חדש למשחק.

לאחר מכן מתבצע אתחול של זרמי הכתיבה והקריאה:

```
reader = new BinaryReader(client.GetStream());
writer = new BinaryWriter(client.GetStream());
```

עכשיו נותר לכתוב ולקרוא נתונים מהסרבר , וזהו בעצם קטע קוד שמתממש במהלך כל רגע שמשחק האונליין רץ, להלן:

```
while (true)
{

    WriteCharacterData(hostChar);
    ReadAndUpdateCharacter(joinChar);

    Thread.Sleep(10);
}
```

האריה של יוצר המשחק שולחת נתונים למשחק השני, האריה של המצטרף מקבל מידע ממנו מהמשחק השני ובכך נשמר ההליך הנורמטיבי של המשחק. תפקיד ה - Thread הוא ליצור הפסקות רגעיות כדי לאפשר למידע לעבור בצורה מסודרת ובטוחה.

הצטרפות למשחק אונליין

לאחר ששחקן כלשהו , פתח סרבר על פי ip ו - port מסוימים, הוא נכנס למצב המתנה שבו הוא מחכה לשחקן נוסף שהתחבר ויאפשר למשחק לפעול. כדי שהשחקן יוכל להתחבר למשחק הכנתי מחלקה שתוכל לטפל בו.

המחלקה JoinOnlineGame:

משתנים –

- **hostIp** – כתובת ה - ip של יוצר המשחק, על מנת להתחבר למשחק.

פונקציות:

- **InitCars** – פונקציה זו מאתחלת את האריות במשחק בצורה דומה לזו שמבוצעת ב – HostOnlineGame. ההבדל הוא בכך שכאן האריה בעל יכולת התנועה הוא אריה ה - join ואריה ה - host הוא חסר תנועה.
- **SocketThread** – הפעולה מתחברת למשחק. שולחת וקוראת נתונים מהשרת.

ההתחברות מבוצעת כך:

```
client = new TcpClient();
client.Connect(hostip, port);
```

בעצם כאן אנו מייצרים משתמש חדש לשרת (משתמש tcp) ומחברים אותו למשחק שנפתח על ip , port ספציפיים.
בהמשך הפעולה מבוצעות פעולות שוטפות כגון קריאה ושליחת נתונים באופן דומה לזה שבמחלקת HostOnlineGame.

ביבליוגרפיה

- ספר: מכניקה ניוטונית של עדי רוזן

- אתר: ריימירס מדריכים ל - XNA - http://riemers.net/

- אתר: http://www.finalfantasykingdom.net/lkmaps.php

- אתר: MSDN – http://msdn.microsoft.com/en-us/

- אתר: ויקיפדיה - http://www.wikipedia.org/

- אתר: אנימציות של הדמויות במלך האריות - http://www.spriters-resource.com/snes/lionking/

- אתר: הורדת תוכנה (Format Factory) להמרת פורמטים של קבצי אודיו - http://www.pcfreetime.com/

- אתר: אפקטי סאונד - http://www.freesfx.co.uk/sfx

Animal.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using System.Diagnostics;
using System.Threading;

namespace xxx
{
    class Animal : Animation
    {
        #region Variables

        public AnimalType AnimalType;
        float NumOfSeconds;
        const float gravity = 9.8f;
        float RightRunSpeed;
        float LeftRunSpeed;
        public float jumpspeed = 0; // jumpspeed to see how fast it
jumps
        public bool jumping; // Is the character jumping?
        public bool falling;
        bool GeneralFalling;
        public bool Hanging;
        public bool Swinging;
        bool PressedFire;
        bool PressedPunch;
        bool PressedDirectionToClimb;
        public bool LeftRunning;
        public bool RightRunning;
        public bool LeftStanding;
        public bool RightStanding;
        bool WasJumping;
```

```
        bool WasFalling;
        bool Crouching;
        bool RunningJump;
        bool Rolling;
        public bool DoubleSlashing;
        public bool ThrowingAway;
        int DelayBetweenFires;
        int DelayBetweenSlashes;
        public BaseKeys baseKeys;
        public AttackBall fireball;
        public AttackBall iceball;
        public SpriteEffects effect;
        SpriteEffects SimabaEffectBeforeThrowing;
        int IndexOfCreatureThatWasThrown;
        float StartFalling_Y;
        float EndFalling_Y;

        #endregion

        /// <summary>
        /// Initializing the parameters
        /// </summary>
        /// <param name="folder"></param>
        /// <param name="state"></param>
        /// <param name="sb"></param>
        /// <param name="position"></param>
        /// <param name="sourceRectangle"></param>
        /// <param name="color"></param>
        /// <param name="rotation"></param>
        /// <param name="origin"></param>
        /// <param name="scale"></param>
        /// <param name="effects"></param>
        /// <param name="layerDepth"></param>
        /// <param name="animalType"></param>
        #region Ctor

        public Animal(Folders folder, States state, SpriteBatch sb,
Vector2 position, Rectangle? sourceRectangle,
            Color color, float rotation, Vector2 origin, Vector2
scale, SpriteEffects effects,
            float layerDepth, AnimalType animalType)
            : base(folder, state, sb, position, sourceRectangle,
color, rotation, origin, scale, effects, layerDepth)
        {
            Game1.UPDATE_EVENT += this.Update;
            Game1.DRAW_EVENT += this.DrawAnimal;
```

```
            this.AnimalType = animalType;

            if (this.AnimalType == AnimalType.Adult_Simba ||
this.AnimalType == AnimalType.Cub_Simba)
            {
                if (Level.LevelNumber == 1)
                {
                    LeftStanding = false;
                    RightStanding = true;
                    this.effects = SpriteEffects.None;
                }

                if (Level.LevelNumber == 2)
                {
                    LeftStanding = true;
                    RightStanding = false;
                    this.effects = SpriteEffects.FlipHorizontally;
                }
            }

            else
            {
                LeftStanding = true;
                RightStanding = false;
                this.effects = SpriteEffects.FlipHorizontally;
            }

            jumping = false;
            falling = false;
            GeneralFalling = false;
            Hanging = false;
            Swinging = false;
            Crouching = false;
            DoubleSlashing = false;
            ThrowingAway = false;
            LeftRunning = false;
            RightRunning = false;
            PressedFire = false;
            PressedPunch = false;
            PressedDirectionToClimb = false;

            if (this.AnimalType == AnimalType.Adult_Simba ||
this.AnimalType == AnimalType.Cub_Simba)
            {
                RightRunSpeed = 9f;
```

```
            LeftRunSpeed = -9f;
        }
        else if (this.AnimalType == AnimalType.lizard ||
            this.AnimalType == AnimalType.hedgehog ||
            this.AnimalType == AnimalType.beetle)
        {
            RightRunSpeed = 2f;
            LeftRunSpeed = -2f;
        }
        //else if (this.AnimalType == AnimalType.lion)
        //{
        //    if (this.jumping)
        //    {
        //        RightRunSpeed = 9f;
        //        LeftRunSpeed = -9f;
        //    }
        //    else
        //    {
        //        RightRunSpeed = 5f;
        //        LeftRunSpeed = -5f;
        //    }
        //}

        jumpspeed = 0;
        DelayBetweenFires = 45;
        DelayBetweenSlashes = 20;

        StartFalling_Y = 0;
        EndFalling_Y = 0;

        WasJumping = false;
        WasFalling = false;
        RunningJump = false;
        Rolling = false;

        IndexOfCreatureThatWasThrown = -1;
        SimabaEffectBeforeThrowing = SpriteEffects.None;

        NumOfSeconds = 1;
    }

    #endregion

    /// <summary>
    /// Updating all the Characters' data
    /// </summary>
```

```
        public void Update()
        {
            if (this.AnimalType == AnimalType.Cub_Simba)
                Console.WriteLine("x: " + this.Pos.X + "  y: " +
this.Pos.Y);

            if (!Game1.IsPaused)
            {
                if (this.AnimalType == AnimalType.lion)
                {
                    if (this.jumping)
                    {
                        RightRunSpeed = 6f;
                        LeftRunSpeed = -6f;
                    }
                    else
                    {
                        RightRunSpeed = 5f;
                        LeftRunSpeed = -5f;
                    }
                }

                PrevState = state;

                #region Borders determination

                if (Level.LevelNumber == 1)
                {
                    if (Pos.X < 200)
                    {
                        Pos = new Vector2(200, Pos.Y);
                    }

                    if (Pos.X > 3900)
                    {
                        Pos = new Vector2(3900, Pos.Y);
                    }

                    if (Pos.Y > 3170)
                    {
                        Pos = new Vector2(Pos.X, 3170);
                    }

                    if (Pos.Y <= 50)
                    {
                        Pos = new Vector2(Pos.X, 50);
```

```
                }
            }

            if (Level.LevelNumber == 2)
            {
                if (Pos.X < 200)
                {
                    Pos = new Vector2(200, Pos.Y);
                }

                if (Pos.X > 3900)
                {
                    Pos = new Vector2(3900, Pos.Y);
                }

                if (Pos.Y > 4000)
                {
                    Pos = new Vector2(Pos.X, 4000);
                }

                if (Pos.Y <= 100)
                {
                    Pos = new Vector2(Pos.X, 100);
                }
            }

            #endregion

            #region If some animation was over

            if (IsAnimationOver)
            {
                this.state = States.Stand;
            }

            #endregion

            #region If_Simba_Is_Blocked

            if ((Map.Locations[(int)(Pos.Y / S.MapsScale),
(int)((Pos.X + 15f) / S.MapsScale)] == GroundType.Blocked ||
                Map.Locations[(int)(Pos.Y / S.MapsScale),
(int)((Pos.X - 15f) / S.MapsScale)] == GroundType.Blocked))
            {
                if (baseKeys.LeftKey() && this.effects ==
SpriteEffects.None)
```

```
                    {
                        this.Pos += new Vector2(-9f, 0);
                    }

                    if (baseKeys.RightKey() && this.effects ==
SpriteEffects.FlipHorizontally)
                    {
                        this.Pos += new Vector2(9f, 0);
                    }
                }

                if (Map.Locations[(int)((Pos.Y + 7f) / S.MapsScale),
(int)(Pos.X / S.MapsScale)] == GroundType.Blocked)
                {
                    if (baseKeys.LeftKey() && this.effects ==
SpriteEffects.FlipHorizontally)
                    {
                        this.Pos = new Vector2(this.Pos.X + 7f,
this.Pos.Y - 7f);
                    }

                    if (baseKeys.RightKey() && this.effects ==
SpriteEffects.None)
                    {
                        this.Pos = new Vector2(this.Pos.X - 7f,
this.Pos.Y - 7f);
                    }
                }

                if (Map.Locations[(int)((Pos.Y - 50f) / S.MapsScale),
(int)(Pos.X / S.MapsScale)] == GroundType.Blocked)
                {
                    this.Pos += new Vector2(0, 50f);
                }

                #endregion

                #region Movement

                if ((baseKeys.RightKey() || baseKeys.LeftKey()) &&
!baseKeys.FireBallKey() && !baseKeys.IceBallKey() &&
                    !Hanging && !Swinging && !Rolling && !Crouching &&
this.state != States.GettingHurt && this.state != States.AfterFalling
&&
                    this.state != States.Slash && this.state !=
States.DoubleSlash && this.state != States.CrouchSlash &&
```

```
                    this.state != States.ThrowingEnemy && this.state
!= States.Roar && this.state != States.Pouncing &&
                    ((this.effects == SpriteEffects.None &&
                    Map.Locations[(int)(Pos.Y / S.MapsScale),
(int)((Pos.X + 15f) / S.MapsScale)] != GroundType.Blocked) ||
                    (this.effects == SpriteEffects.FlipHorizontally &&
                    Map.Locations[(int)(Pos.Y / S.MapsScale),
(int)((Pos.X - 15f) / S.MapsScale)] != GroundType.Blocked)))
                {
                    if (baseKeys.RightKey())
                    {
                        RightRunning = true;
                        LeftStanding = false;
                        this.effects = SpriteEffects.None;
                        Pos += new Vector2(RightRunSpeed, 0);
                    }

                    if (baseKeys.LeftKey())
                    {
                        LeftRunning = true;
                        RightStanding = false;
                        this.effects = SpriteEffects.FlipHorizontally;
                        Pos += new Vector2(LeftRunSpeed, 0);
                    }

                    if (jumping)
                    {
                        RunningJump = true;
                    }
                    if (this.AnimalType == AnimalType.Cub_Simba &&
baseKeys.DownKey()) // When cub
                    {
                        Rolling = true;
                    }
                    else
                    {
                        this.state = States.Running;
                    }
                }

                if (RunningJump)
                {
                    this.state = States.Running_Jump;
                    RunningJump = false;
                }
```

```
                // ---------------------------------------------------
----------------------------
                // This condition is in order to avoid the falling
from platform while running jump
                if (this.state == States.Running_Jump &&
Map.Locations[(int)((Pos.Y + 5f) / S.MapsScale),
                    (int)(Pos.X / S.MapsScale)] ==
GroundType.Platform)
                {
                    this.Pos = new Vector2(this.Pos.X, this.Pos.Y +
5f);
                    this.state = States.Stand;
                    jumping = false;
                }
                // ---------------------------------------------------
----------------------------

                if (Rolling)
                {
                    this.state = States.Rolling;
                    Rolling = false;
                }

                if (!baseKeys.RightKey() && RightRunning && this.state
!= States.GettingHurt)
                {
                    RightRunning = false;
                    RightStanding = true;

                    if (!Hanging && Map.Locations[(int)(Pos.Y /
S.MapsScale), (int)(Pos.X / S.MapsScale)] != GroundType.Air)
                    {
                        this.state = States.Stand;
                    }
                }

                if (!baseKeys.LeftKey() && LeftRunning && this.state
!= States.GettingHurt)
                {
                    LeftRunning = false;
                    LeftStanding = true;

                    if (!Hanging && Map.Locations[(int)(Pos.Y /
S.MapsScale), (int)(Pos.X / S.MapsScale)] != GroundType.Air)
                    {
                        this.state = States.Stand;
```

```
                    }
                }

                #endregion

                #region Simaba's_Punches (When adult)

                if (this.AnimalType == AnimalType.Adult_Simba)
                {

                    if (DelayBetweenSlashes < 20)
                    {
                        DelayBetweenSlashes += 1;
                    }

                    if (!jumping && !PressedPunch && !Hanging &&
!Crouching && this.state != States.Roar &&
                        this.state != States.ThrowingEnemy &&
this.state != States.GettingHurt && baseKeys.PunchKey())
                    {
                        PressedPunch = true;

                        if (DelayBetweenSlashes == 20)
                        {
                            DelayBetweenSlashes = 0;
                            this.state = States.Slash;
                        }
                    }

                    if (baseKeys.ShiftKey() && PressedPunch &&
!DoubleSlashing && !jumping)
                    {
                        for (int i = 1; i < Level.Characters.Count;
i++)
                        {
                            if (!ThrowingAway && ((this.Pos.X -
Level.Characters[i].Pos.X <= 200f &&
                                this.Pos.X - Level.Characters[i].Pos.X
>= 150f &&
                                Math.Abs(this.Pos.Y -
Level.Characters[i].Pos.Y) <= 2f &&
                                this.effects ==
SpriteEffects.FlipHorizontally &&
                                Level.Characters[i].effects ==
SpriteEffects.None) ||
```

```
                                (Level.Characters[i].Pos.X -
this.Pos.X <= 200f &&
                                Level.Characters[i].Pos.X - this.Pos.X
>= 150f &&
                                Math.Abs(this.Pos.Y -
Level.Characters[i].Pos.Y) <= 2f &&
                                this.effects == SpriteEffects.None &&
                                Level.Characters[i].effects ==
SpriteEffects.FlipHorizontally)))
                            {
                                ThrowingAway = true;
                                IndexOfCreatureThatWasThrown = i;
                                SimabaEffectBeforeThrowing =
this.effects;
                                Level.Characters[i].state =
States.TossedBySimba;

                                if (this.effects ==
SpriteEffects.None)
                                {
                                    this.Pos = Level.Characters[i].Pos
+ new Vector2(-195f, 0);
                                }

                                else if (this.effects ==
SpriteEffects.FlipHorizontally)
                                {
                                    this.Pos = Level.Characters[i].Pos
+ new Vector2(195f, 0);
                                }

                                break;
                            }

                            else
                            {
                                DoubleSlashing = true;
                                this.state = States.DoubleSlash;
                            }
                        }
                    }

                    if (!baseKeys.PunchKey() && PressedPunch)
                    {
                        PressedPunch = false;
                        DoubleSlashing = false;
```

```
                    }
                }

                #endregion

                #region Crouch + CrouchSlash

                if ((baseKeys.DownKey() && !baseKeys.RightKey() &&
!baseKeys.LeftKey() && !Crouching && !falling && !jumping &&
                    this.state != States.Hanging && this.state !=
States.Climb &&
                    this.state != States.DoubleSlash && this.state !=
States.Slash
                    && this.state != States.GettingHurt && this.state
!= States.GettingSlashed
                    && this.state != States.AfterFalling && this.state
!= States.ThrowingEnemy && this.state != States.Roar &&
                    Map.Locations[(int)(Pos.Y / S.MapsScale),
(int)(Pos.X / S.MapsScale)] != GroundType.Air)
                    || (IsAnimationOver && Crouching))
                {
                    Crouching = true;
                    this.state = States.Crouch;
                }

                if (baseKeys.PunchKey() && !PressedPunch && Crouching)
                {
                    PressedPunch = true;

                    if (DelayBetweenSlashes == 20)
                    {
                        DelayBetweenSlashes = 0;
                        this.state = States.CrouchSlash;
                    }
                }

                if (!baseKeys.DownKey() && Crouching && this.state !=
States.CrouchSlash)
                {
                    this.state = States.Stand;
                    Crouching = false;
                }

                #endregion

                #region FireBalls & IceBalls
```

```
                if (DelayBetweenFires < 45)
                {
                    DelayBetweenFires += 1;
                }

                if ((baseKeys.FireBallKey() || baseKeys.IceBallKey())
&& !PressedFire && !Hanging &&
                    !baseKeys.RightKey() && !baseKeys.LeftKey() &&
!Crouching && !ThrowingAway && !jumping &&
                    this.state != States.Slash && this.state !=
States.DoubleSlash && this.state != States.AfterFalling)
                {
                    PressedFire = true;

                    if (DelayBetweenFires == 45)
                    {
                        DelayBetweenFires = 0;

                        if (baseKeys.FireBallKey())
                        {
                            //LionSounds.fireballsound.Play();

                            // יריית אפקט ניגון תחילת בין מסוים דילאי שיהיה רוצה אני
השאגה אנימציית תחילת לבין האש כדור
                            // לשמיעה הראייה בין סנכרון שיהיה כדי
                            Thread.Sleep(25);

                            ShootFireBall();
                        }

                        if (baseKeys.IceBallKey())
                        {
                            ShootIceBall();
                        }
                    }
                }

                if (!baseKeys.FireBallKey() && !baseKeys.IceBallKey())
                {
                    PressedFire = false;
                }

                #endregion

                #region Jump
```

```
                if (baseKeys.UpKey() && !baseKeys.FireBallKey() &&
!baseKeys.IceBallKey() && !jumping &&
                    this.state != States.DoubleSlash && this.state !=
States.Slash && this.state != States.GettingSlashed &&
                    this.state != States.GettingHurt && this.state !=
States.CrouchSlash && this.state != States.Climb &&
                    this.state != States.ThrowingEnemy && this.state
!= States.AfterFalling &&
                    Map.Locations[(int)(Pos.Y / S.MapsScale),
(int)(Pos.X / S.MapsScale)] != GroundType.Air)
                {
                    jumping = true;
                    WasJumping = true;
                    StartFalling_Y = this.Pos.Y / S.MapsScale;
                    jumpspeed = -16.0f; // מעלה כלפי דחיפה לו נותן
                    this.state = States.Jump;
                }

                if (jumping)
                {
                    Pos += new Vector2(0, jumpspeed);

                    if (jumpspeed <= 25f) // מהירות הגבלת
                    {
                        jumpspeed += 0.6f; // הקפיצה תהיה קצב באיזה
                    }

                    if (this.AnimalType == AnimalType.Cub_Simba ||
this.AnimalType == AnimalType.Adult_Simba)
                    {
                        foreach (Vector2 hangingData in
Map.HangLocations)
                        {
                            if (!Hanging && jumpspeed >= 0 &&
                                Math.Abs((new Vector2(Pos.X /
S.MapsScale, Pos.Y / S.MapsScale)
                                - new Vector2(hangingData.X,
hangingData.Y)).Length()) <= 15f &&
                                ((this.effects ==
SpriteEffects.FlipHorizontally &&
                                Map.Locations[(int)(hangingData.Y),
                                (int)((hangingData.X - 5f))] ==
GroundType.Platform) ||
                                (this.effects == SpriteEffects.None &&
                                Map.Locations[(int)(hangingData.Y),
```

```
                                (int)((hangingData.X + 5f))] ==
GroundType.Platform)))
                            {
                                Hanging = true;
                                jumping = false;
                                this.state = States.Hanging;
                                this.Pos = new Vector2(
                                    hangingData.X * S.MapsScale,
                                    hangingData.Y * S.MapsScale);

                                break;
                            }
                        }

                        if (this.AnimalType ==
xxx.AnimalType.Adult_Simba)
                        {
                            foreach (Vector2 SwingData in
Map.SwingLocations)
                            {
                                if (!Swinging && jumpspeed >= 0 &&
                                    Math.Abs((new Vector2(Pos.X /
S.MapsScale, Pos.Y / S.MapsScale)
                                    - new Vector2(SwingData.X,
SwingData.Y)).Length()) <= 25f &&
                                    ((this.effects ==
SpriteEffects.FlipHorizontally &&
                                    Map.Locations[(int)(SwingData.Y),
                                    (int)((SwingData.X - 5f))] ==
GroundType.Air) ||
                                    (this.effects ==
SpriteEffects.None &&
                                    Map.Locations[(int)(SwingData.Y),
                                    (int)((SwingData.X + 5f))] ==
GroundType.Air)))
                                {
                                    Swinging = true;
                                    jumping = false;
                                    this.state = States.Swinging;
                                    this.Pos = new Vector2(
                                        SwingData.X * S.MapsScale,
                                        SwingData.Y * S.MapsScale);

                                    break;
                                }
                            }
```

```
                        }
                    }
                }

                if (Map.Locations[(int)(Pos.Y / S.MapsScale),
(int)(Pos.X / S.MapsScale)] == GroundType.Air || falling)
                {
                    for (int i = 0; i < Math.Abs(jumpspeed); i++)
                    {
                        if (Map.Locations[(int)((Pos.Y + i) /
S.MapsScale), (int)(Pos.X / S.MapsScale)] == GroundType.Platform)
                        {
                            jumpspeed = i;
                            Pos += new Vector2(0, jumpspeed); // לגרום
שטוח משטח על מעלה לעלות לדמות
                            jumping = false;
                            falling = false;
                            EndFalling_Y = this.Pos.Y / S.MapsScale;

                            if ((this.AnimalType ==
AnimalType.Cub_Simba || this.AnimalType == AnimalType.Adult_Simba) &&
                                (WasFalling || WasJumping) &&
EndFalling_Y - StartFalling_Y >= 200f)
                            {
                                this.state = States.AfterFalling;
                            }

                            else
                            {
                                this.state = States.Stand;
                            }

                            WasJumping = false;
                            WasFalling = false;

                            break;
                        }

                        else
                        {
                            for (int k = 0; k < 10; k++)
                            {
                                if (Map.Locations[(int)((Pos.Y - k) /
S.MapScale),
                                    (int)(Pos.X / S.MapsScale)] ==
GroundType.Platform
```

```
                                    && !jumping)
                                {
                                    this.Pos = new Vector2(Pos.X,
Pos.Y - k);
                                }
                            }

                            if (falling) // מתלייה נפילה
                            {
                                falling = false;
                                WasFalling = true;
                                GeneralFalling = true;
                            }
                        }
                    }
                }


                #endregion

                #region Hanging + Climbing

                if (Hanging)
                {
                    if (baseKeys.LeftKey() && this.effects ==
SpriteEffects.FlipHorizontally)
                    {
                        PressedDirectionToClimb = true;
                    }

                    if (baseKeys.RightKey() && this.effects ==
SpriteEffects.None)
                    {
                        PressedDirectionToClimb = true;
                    }
                }

                if (PressedDirectionToClimb)
                {
                    this.state = States.Climb;

                    PressedDirectionToClimb = false;
                }

                if ((jumping) || (!PressedDirectionToClimb &&
IsAnimationOver && Hanging))
```

```
                {
                    Hanging = false;
                }

                #endregion

                #region Swinging

                if (Swinging)
                {
                    //this.Pos = Vector2.Transform(this.Pos,
Matrix.CreateRotationX(360));

                    if (jumping)
                    {
                        this.state = States.Running_Jump;
                        Swinging = false;
                    }
                }

                #endregion

                #region Throwing_Away

                if (ThrowingAway)
                {
                    DelayBetweenSlashes = 0;
                    this.state = States.ThrowingEnemy;

                    if (IsAnimationOver && SimabaEffectBeforeThrowing
== SpriteEffects.FlipHorizontally)
                    {
                        ThrowingAway = false;
                        LeftStanding = false;
                        LeftRunning = false;
                        RightStanding = true;
                        this.effects = SpriteEffects.None;

Level.Characters[IndexOfCreatureThatWasThrown].effects =
SpriteEffects.FlipHorizontally;
                    }

                    else if (IsAnimationOver &&
SimabaEffectBeforeThrowing == SpriteEffects.None)
                    {
                        ThrowingAway = false;
```

```
                        RightStanding = false;
                        RightRunning = false;
                        LeftStanding = true;
                        this.effects = SpriteEffects.FlipHorizontally;

Level.Characters[IndexOfCreatureThatWasThrown].effects =
SpriteEffects.None;
                    }
                }

                #endregion

                #region General falling & After hanging falling

                if (baseKeys.DownKey() && (this.state ==
States.Hanging || this.state == States.Swinging) && !falling &&
!Crouching)
                {
                    falling = true;
                    WasFalling = true;
                    Hanging = false;
                    Swinging = false;
                    jumpspeed = -1.1f;
                    StartFalling_Y = this.Pos.Y / S.MapsScale;
                    this.state = States.Falling;
                }

                if (this.state == States.Falling && EndFalling_Y -
StartFalling_Y < 200f &&
                    Map.Locations[(int)(Pos.Y / S.MapsScale),
(int)(Pos.X / S.MapsScale)] == GroundType.Platform)
                {
                    this.state = States.Stand;
                }

                if (Map.Locations[(int)(Pos.Y / S.MapsScale),
(int)(Pos.X / S.MapsScale)] == GroundType.Air &&
                    !jumping && !GeneralFalling)
                {
                    GeneralFalling = true;
                }
                if (GeneralFalling)
                {
                    Fall();
                    GeneralFalling = false;
                }
```

```
                else
                {
                    NumOfSeconds = 0;
                }

                #endregion

                #region Collision

                if (this.AnimalType == AnimalType.Adult_Simba ||
this.AnimalType == AnimalType.Cub_Simba) // Collision with simba
                {
                    for (int i = 0; i < Level.Characters.Count; i++)
                    {
                        Collision.CheckCollision(this, this.index,
Level.Characters[i], Level.Characters[i].index);
                    }
                }

                #endregion
            }
        }

        /// <summary>
        /// Generating fireball
        /// </summary>
        public void ShootFireBall()
        {
            this.state = States.Roar;

            //if (this.effects == SpriteEffects.None)
            //{
            //    effect = SpriteEffects.None;
            //    fireball = new AttackBall(S.spb,
TheDict.AttacksDic[Attacks.fireball],
            //        new Vector2(Pos.X + 50, Pos.Y - 70), 0f, 14,
100, effect);
            //}

            //if (this.effects == SpriteEffects.FlipHorizontally)
            //{
            //    effect = SpriteEffects.FlipHorizontally;
            //    fireball = new AttackBall(S.spb,
TheDict.AttacksDic[Attacks.fireball],
            //        new Vector2(Pos.X - 80, Pos.Y - 70), 0f, -14,
100, effect);
```

```
            //}
        }

        /// <summary>
        /// Generating iceball
        /// </summary>
        public void ShootIceBall()
        {
            this.state = States.Roar;

            if (this.effects == SpriteEffects.None)
            {
                effect = SpriteEffects.None;
                iceball = new AttackBall(S.spb,
TheDict.AttacksDic[Attacks.iceball],
                    new Vector2(Pos.X + 30, Pos.Y - 102), 0f, 14, 100,
effect);
            }

            if (this.effects == SpriteEffects.FlipHorizontally)
            {
                effect = SpriteEffects.FlipHorizontally;
                iceball = new AttackBall(S.spb,
TheDict.AttacksDic[Attacks.iceball],
                    new Vector2(Pos.X - 122, Pos.Y - 102), 0f, -14,
100, effect);
            }
        }

        /// <summary>
        /// Drawing the character
        /// </summary>
        public void DrawAnimal()
        {
            base.DrawObject();
        }

        /// <summary>
        /// Falling, according the physics rules
        /// </summary>
        public void Fall()
        {
            // כעבור מושגת המקסימלית שהמהירות היא התנאי משמעות - מהירות הגבלת זו
וחצי שנייה
            if (NumOfSeconds / 60 <= 1.5f)
            {
```

```
                NumOfSeconds++;
            }

            // התחלתית מהירות זה הזה 11 ה
            Pos += new Vector2(0, 11f + 0.5f * gravity *
((NumOfSeconds * NumOfSeconds) / 3600f));
        }
    }
}
```

Animation.cs:

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

namespace xxx
{
    class Animation : SpriteObject
    {
        public Folders folder { get; set; }
        public States state { get; set; }
        public States PrevState;
        public bool IsAnimationOver;
        public int index;
        int indexSlow;
        int slow;

        /// <summary>
        /// Initializing the parameters
        /// </summary>
        /// <param name="folder"></param>
        /// <param name="state"></param>
        /// <param name="sb"></param>
        /// <param name="position"></param>
        /// <param name="sourceRectangle"></param>
        /// <param name="color"></param>
        /// <param name="rotation"></param>
        /// <param name="origin"></param>
        /// <param name="scale"></param>
        /// <param name="effects"></param>
        /// <param name="layerDepth"></param>
        #region Ctor

        public Animation(Folders folder, States state, SpriteBatch sb,
Vector2 position, Rectangle? sourceRectangle, Color color, float
```

```
rotation, Vector2 origin, Vector2 scale, SpriteEffects effects, float
layerDepth)
            : base(null, position, sourceRectangle, color, rotation,
origin, scale, effects, layerDepth)
        {
            IsAnimationOver = false;
            this.folder = folder;
            this.state = state;
            this.indexSlow = 0;
            this.index = 0;
            this.slow = 0;
        }

        #endregion

        /// <summary>
        /// Drawing a texture
        /// </summary>
        public override void DrawObject()
        {
            Page p = TheDict.dic[folder][state];
            base.texture = p.tex;
            base.sourceRectangle = p.rec[index % p.rec.Count];

            try
            {
                if (this.effects == SpriteEffects.None)
                {
                    base.origin = p.org[index % p.rec.Count];
                }

                if (this.effects == SpriteEffects.FlipHorizontally)
                {
                    base.origin = p.FlippedOrg[index % p.rec.Count];
                }
            }

            catch { }

            base.DrawObject();

            this.slow = Page.DefineStatesSlows(this.state);

            if (PrevState != state)
            {
                IsAnimationOver = false;
```

```
                indexSlow = 0;
                index = 0; // מהפריים נתחיל, לאחרת אחת מאנימציה שבמעבר כדי
אנימציה סטריפ בכל הראשון
            }

            if (indexSlow == this.slow && !Game1.IsPaused)
            {
                index++;

                if (this.state == States.Running_Jump && index == 8 &&
                    Map.Locations[(int)(this.Pos.Y / S.MapsScale),
(int)(this.Pos.X / S.MapsScale)] == GroundType.Air)
                {
                    index -= 1;
                }

                if (this.state == States.Crouch && index == 4)
                {
                    index -= 1;
                }

                if (this.state == States.ThrowingEnemy && index >= 1
&& index <= 4)
                {
                    if (this.effects == SpriteEffects.None)
                    {
                        this.Pos += new Vector2(20f, 0);
                    }

                    if (this.effects ==
SpriteEffects.FlipHorizontally)
                    {
                        this.Pos += new Vector2(-20f, 0);
                    }
                }

                if (this.state == States.TossedBySimba)
                {
                    if (index >= 5 && index <= 13)
                    {
                        this.color = Color.Transparent;
                    }

                    else
                    {
                        this.color = Color.White;
```

```
                    }

                    if (index > 6)
                    {
                        if (this.effects == SpriteEffects.None)
                        {
                            this.Pos += new Vector2(20, 0);
                        }

                        else if (this.effects ==
SpriteEffects.FlipHorizontally)
                        {
                            this.Pos += new Vector2(-20f, 0);
                        }
                    }
                }

                if (this.state == States.Climb)
                {
                    if (index >= 3 && index <= 7)
                    {
                        if (Level.hero.AnimalType ==
AnimalType.Cub_Simba)
                        {
                            if (this.effects == SpriteEffects.None)
                            {
                                this.Pos += new Vector2(19f, 0);
                            }

                            if (this.effects ==
SpriteEffects.FlipHorizontally)
                            {
                                this.Pos += new Vector2(-19f, 0);
                            }
                        }

                        if (Level.hero.AnimalType ==
AnimalType.Adult_Simba)
                        {
                            if (this.effects == SpriteEffects.None)
                            {
                                this.Pos += new Vector2(10f, 0);
                            }

                            if (this.effects ==
SpriteEffects.FlipHorizontally)
```

```
                            {
                                this.Pos += new Vector2(-10f, 0);
                            }
                        }
                    }
                }

                if (this.state == States.DoubleSlash || this.state ==
States.AfterFalling)
                {
                    if (index == p.org.Count)
                    {
                        IsAnimationOver = true;
                    }
                }

                else
                {
                    if (index + 1 == p.org.Count)
                    {
                        IsAnimationOver = true;
                    }
                }

                indexSlow = 0;
            }

            if (indexSlow > slow)
            {
                indexSlow = 0;
            }

            indexSlow++;
        }
    }
}
```

AttackBall.cs:

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

namespace xxx
{
    class AttackBall : SpriteObject
    {
        public int age;
        public int speed;
        public int damage;
        public Vector2 velocity;

        /// <summary>
        /// Initializing the parameters
        /// </summary>
        /// <param name="sb"></param>
        /// <param name="texture"></param>
        /// <param name="position"></param>
        /// <param name="rotation"></param>
        /// <param name="speed"></param>
        /// <param name="damage"></param>
        /// <param name="effect"></param>
        #region Ctor

        public AttackBall(SpriteBatch sb, Texture2D texture, Vector2
position, float rotation, int speed, int damage, SpriteEffects effect)
            : base(texture, position, null, Color.White, 0f, new
Vector2(1f, 1f), new Vector2(0.5f, 0.5f), SpriteEffects.None, 0)
        {
            Game1.UPDATE_EVENT += this.Update;
            Game1.DRAW_EVENT += this.Draw;

            this.age = 0;
            this.damage = damage;
```

```
            this.speed = speed;
            velocity = new Vector2(speed, 0);

            // BASE האפקט מחדל שברירת מפני הם הללו התנאים שני NONE מ ההורשה בגלל
כ נקבע
            // זה מצב לשנות רוצה ואני
            if (effect == SpriteEffects.None)
            {
                effects = SpriteEffects.None;
            }

            if (effect == SpriteEffects.FlipHorizontally)
            {
                effects = SpriteEffects.FlipHorizontally;
            }
        }

        #endregion

        //public void Kill()
        //{
        //    this.age = 200;
        //}

        //public int Damage
        //{
        //    get { return damage; }
        //}

        //public bool IsDead()
        //{
        //    return (this.age > 100);
        //}

        /// <summary>
        /// Updateing the attack ball's data
        /// </summary>
        public void Update()
        {
            if (age >= 60)
            {
                Game1.DRAW_EVENT -= this.Draw;
                Game1.DRAW_EVENT -= this.Update;
            }

            else
```

```
            {
                age++;
                Pos += velocity;
            }

        }

        /// <summary>
        /// Drawing the attack ball
        /// </summary>
        public void Draw()
        {
            base.DrawObject();
        }
    }
}
```

BaseKeys.cs:

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

namespace xxx
{
    abstract class BaseKeys
    {
        public abstract bool UpKey();
        public abstract bool DownKey();
        public abstract bool LeftKey();
        public abstract bool RightKey();
        public abstract bool ShiftKey();
        public abstract bool PunchKey();
        public abstract bool FireBallKey();
        public abstract bool IceBallKey();
    }
}
```

BotKeys.cs:

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

namespace xxx
{
    class BotKeys : BaseKeys
    {
        Animal animal;
        Animal Simba;
        bool up, down, right, left, punch, shift, iceball, fireball;
        bool TurnRight, TurnLeft;
        float Start_X;
        float DistanceBetweenSimbaAndBigEnemy;
        bool SimbaIsAboveEnemy;
        bool SimbaAndEnemyIsInTheSameHight;
        bool SimbaIsUnderEnemy;
        bool FirstCheck = true;
        bool WasRight = false;
        bool WasLeft = false;
        bool stop = false;

        /// <summary>
        /// Initializing the parameters
        /// </summary>
        /// <param name="lion"></param>
        /// <param name="Animal_Type"></param>
        public BotKeys(Animal animal, AnimalType Animal_Type)
        {
            this.Simba = Level.hero;
            this.animal = animal;
            up = down = right = left = punch = shift = iceball =
fireball = false;
            TurnRight = true;
            TurnLeft = false;
```

```
        Start_X = this.animal.Pos.X;

        SimbaIsAboveEnemy = false;
        SimbaIsUnderEnemy = false;
        SimbaAndEnemyIsInTheSameHight = false;

        Game1.UPDATE_EVENT += this.Update;
    }

    /// <summary>
    ///
    /// </summary>
    /// <returns></returns>
    public override bool UpKey()
    {
        return up;
    }

    /// <summary>
    ///
    /// </summary>
    /// <returns></returns>
    public override bool DownKey()
    {
        return down;
    }

    /// <summary>
    ///
    /// </summary>
    /// <returns></returns>
    public override bool RightKey()
    {
        return right;
    }

    /// <summary>
    ///
    /// </summary>
    /// <returns></returns>
    public override bool LeftKey()
    {
        return left;
    }
```

```
/// <summary>
///
/// </summary>
/// <returns></returns>
public override bool ShiftKey()
{
    return shift;
}

/// <summary>
///
/// </summary>
/// <returns></returns>
public override bool PunchKey()
{
    return punch;
}

/// <summary>
///
/// </summary>
/// <returns></returns>
public override bool FireBallKey()
{
    return fireball;
}

/// <summary>
///
/// </summary>
/// <returns></returns>
public override bool IceBallKey()
{
    return iceball;
}

/// <summary>
/// Updateing AI reaction regarding to the hero
/// </summary>
public void Update()
{
    #region Lirzard + Hedgehog + Beetle Movement

    if (this.animal.AnimalType == AnimalType.lizard)
    {
```

```
                if ((this.animal.effects ==
SpriteEffects.FlipHorizontally
                     && this.animal.Pos.X > this.Simba.Pos.X &&
this.animal.Pos.Y - this.Simba.Pos.Y <= 2f &&
                     this.animal.Pos.X - this.Simba.Pos.X <= 125f) ||
                    (this.animal.effects == SpriteEffects.None
                    && this.animal.Pos.X < this.Simba.Pos.X &&
this.animal.Pos.Y - this.Simba.Pos.Y <= 2f
                    && this.Simba.Pos.X - this.animal.Pos.X <= 125f))
                {
                    this.animal.state = States.Slash;
                    left = false;
                    right = false;
                }

                if (this.animal.state != States.Slash)
                {
                    Walk();
                }
            }


            if (this.animal.AnimalType == AnimalType.hedgehog ||
this.animal.AnimalType == AnimalType.beetle)
            {
                Walk();
            }

            #endregion

            if (this.animal.AnimalType == AnimalType.lion)
            {
                if (FirstCheck)
                {
                    if (this.animal.Pos.X > this.Simba.Pos.X)
                    {
                        left = true;
                        WasLeft = true;
                        WasRight = false;
                        right = false;
                    }

                    else
                    {
                        right = true;
                        WasRight = true;
```

```
                    WasLeft = false;
                    left = false;
                }

                FirstCheck = false;
            }

            if (this.animal.Pos.Y < this.Simba.Pos.Y) // סקאר אם
גבוה יותר במקום
            {
                SimbaIsUnderEnemy = true;
                SimbaIsAboveEnemy = false;
            }


            else if (this.animal.Pos.Y > this.Simba.Pos.Y) // אם
נמוך יותר במקום סקאר
            {
                SimbaIsAboveEnemy = true;
                SimbaIsUnderEnemy = false;
            }

            else
            {
                SimbaAndEnemyIsInTheSameHight = true;
                SimbaIsAboveEnemy = false;
                SimbaIsUnderEnemy = false;
            }

            if (SimbaAndEnemyIsInTheSameHight)
            {
                if (this.animal.Pos.X < this.Simba.Pos.X &&
this.Simba.Pos.X - this.animal.Pos.X >= 100f &&
                    this.Simba.Pos.Y == this.animal.Pos.Y)
                {
                    right = true;
                    WasRight = true;
                    left = false;
                    WasLeft = false;
                }

                else if (this.animal.Pos.X > this.Simba.Pos.X &&
this.animal.Pos.X - this.Simba.Pos.X >= 100f &&
                    this.Simba.Pos.Y == this.animal.Pos.Y)
                {
                    left = true;
```

```
                        WasLeft = true;
                        right = false;
                        WasRight = false;
                    }

                    else
                    {
                        stop = true;
                        right = false;
                        left = false;
                        up = false;
                    }
                }

                if (Math.Abs(this.Simba.Pos.Y - this.animal.Pos.Y) <=
5f &&
                    ((this.Simba.Pos.X > this.animal.Pos.X &&
this.Simba.Pos.X - this.animal.Pos.X <= 100f) ||
                    (this.Simba.Pos.X < this.animal.Pos.X &&
this.animal.Pos.X - this.Simba.Pos.X <= 100f)))
                {
                    stop = true;
                }

                if (SimbaIsUnderEnemy || SimbaIsAboveEnemy)
                {
                    stop = false;

                    if (WasRight && !stop)
                    {
                        right = true;
                        left = false;
                        WasLeft = false;
                    }

                    else if (WasLeft && !stop)
                    {
                        left = true;
                        right = false;
                        WasRight = false;
                    }

                    // שיש איפה מעל קפיצה יכולת זיהוי בשביל הם 220 ה עם התנאים
אדום
                    // המקסימלי הקפיצה גובה זה 220
```

```
                    if (SimbaIsAboveEnemy)
                    {
                        if ((Map.Locations[(int)(this.animal.Pos.Y /
S.MapsScale),
                                (int)((this.animal.Pos.X - 15f) /
S.MapsScale)] == GroundType.Blocked &&
                            Map.Locations[(int)((this.animal.Pos.Y -
220f) / S.MapsScale),
                                (int)((this.animal.Pos.X - 15f) /
S.MapsScale)] != GroundType.Blocked) ||
                            (Map.Locations[(int)(this.animal.Pos.Y /
S.MapsScale),
                                (int)((this.animal.Pos.X + 15f) /
S.MapsScale)] == GroundType.Blocked &&
                            Map.Locations[(int)((this.animal.Pos.Y -
220f) / S.MapsScale),
                                (int)((this.animal.Pos.X + 15f) /
S.MapsScale)] != GroundType.Blocked))
                        {
                            up = true;
                        }
                        else if (Map.Locations[(int)(this.animal.Pos.Y
/ S.MapsScale),
                                (int)((this.animal.Pos.X - 15f) /
S.MapsScale)] == GroundType.Air ||
                            Map.Locations[(int)(this.animal.Pos.Y /
S.MapsScale),
                                (int)((this.animal.Pos.X + 15f) /
S.MapsScale)] == GroundType.Air)
                        {
                            up = true;
                        }
                        else
                        {
                            up = false;
                        }
                    }

                    // אדום מחסום לעבור מצליח לא שאני למקרה זה שמתחת התנאים שני
גבוה
                    // עד שהלכה ממה הנגדי לכיוון ללכת המלאכותית לבינה סימן זה לכן
עכשיו
                    if ((Map.Locations[(int)(this.animal.Pos.Y /
S.MapsScale),
                          (int)((this.animal.Pos.X + 15f) /
S.MapsScale)] == GroundType.Blocked &&
```

```
                        Map.Locations[(int)((this.animal.Pos.Y - 220f)
/ S.MapsScale),
                           (int)((this.animal.Pos.X + 15f) /
S.MapsScale)] == GroundType.Blocked))
                    {
                        left = true;
                        right = false;
                        WasLeft = true;
                        WasRight = false;
                    }

                    if ((Map.Locations[(int)(this.animal.Pos.Y /
S.MapsScale),
                        (int)((this.animal.Pos.X - 15f) / S.MapsScale)]
== GroundType.Blocked &&
                    Map.Locations[(int)((this.animal.Pos.Y - 220f) /
S.MapsScale),
                        (int)((this.animal.Pos.X - 15f) / S.MapsScale)]
== GroundType.Blocked))
                    {
                        right = true;
                        left = false;
                        WasRight = true;
                        WasLeft = false;
                    }

                    if (SimbaIsUnderEnemy)
                    {
                        if (this.Simba.Pos.X > this.animal.Pos.X &&
this.Simba.Pos.X - this.animal.Pos.X >= 100f)
                        {
                            right = true;
                            left = false;
                            WasRight = true;
                            WasLeft = false;
                        }


                        if (this.Simba.Pos.X < this.animal.Pos.X &&
this.animal.Pos.X - this.Simba.Pos.X >= 100f)
                        {
                            left = true;
                            right = false;
                            WasLeft = true;
                            WasRight = false;
                        }
```

```
                }
            }
        }
    }

    /// <summary>
    /// Walking side to side after passing determined distance
each time
    /// </summary>
    public void Walk()
    {
        if (this.animal.Pos.X < Start_X)
        {
            TurnRight = true;
            TurnLeft = false;
        }

        if (this.animal.Pos.X > Start_X + 550f)
        {
            TurnLeft = true;
            TurnRight = false;
        }

        if (TurnRight)
        {
            right = true;
            left = false;
        }

        if (TurnLeft)
        {
            left = true;
            right = false;
        }
    }
  }
}
```

Camera.cs:

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

namespace xxx
{
    class Camera
    {
        #region Data

        public Matrix Mat { get; private set; }
        public Ifocus Focus { get; private set; }
        public Vector2 Zoom { get; private set; }
        public Viewport View { get; private set; }
        public Vector2 Pos { get; private set; }

        #endregion

        /// <summary>
        /// Initializing the parameters
        /// </summary>
        /// <param name="focus"></param>
        /// <param name="zoom"></param>
        /// <param name="view"></param>
        public Camera(Ifocus focus, Vector2 zoom, Viewport view)
        {
            Game1.UPDATE_EVENT += this.UpdateMat;

            Focus = focus;
            Zoom = zoom;
            View = view;
            Pos = Focus.Pos;
        }

        /// <summary>
```

```
        /// Updating the cam's data
        /// </summary>
        public void UpdateMat()
        {
            Mat = Matrix.CreateTranslation(-Pos.X, -Pos.Y, 0) *
                  Matrix.CreateScale(Zoom.X, Zoom.Y, 1) *
                  Matrix.CreateTranslation(View.Width / 2, View.Height
/ 2, 0);

            if (Level.LevelNumber == 1)
            {
                Pos = Vector2.Lerp(new Vector2(
                    MathHelper.Clamp(Focus.Pos.X, (80 + View.Width /
2), 3350),
                    MathHelper.Clamp(Focus.Pos.Y, -300 + View.Height,
3170 - View.Height / 2)), Pos, 0.8f);
            }

            if (Level.LevelNumber == 2)
            {
                Pos = Vector2.Lerp(new Vector2(
                    MathHelper.Clamp(Focus.Pos.X, (View.Width / 2),
3360),
                    MathHelper.Clamp(Focus.Pos.Y, -2000 + View.Height,
4000 - View.Height / 2)), Pos, 0.8f);
            }
        }
    }
}
```

Circle.cs:

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

namespace xxx
{
    class Circle
    {
        public Vector2 center = Vector2.Zero; // The center of the
Circle
        public float radius; // The radius of the Circle

        /// <summary>
        /// constructor which initiallize those paramaters.
        /// </summary>
        /// <param name="center">The center of the circle</param>
        /// <param name="tex">The texture of the car</param>
        /// <param name="scale">The scale of the car</param>
        /// <param name="org">The origin of the car</param>
        /// <param name="ifBig">Set if this is the Big Circle or
regular Circle</param>
        public Circle(Vector2 center, Rectangle rec, Vector2 scale,
bool ifBig)
        {
            this.center.X = center.X * scale.X;
            this.center.Y = center.Y * scale.Y;
            this.radius = find_radius(rec, scale, ifBig);
        }

        /// <summary>
        /// Find the center center of the car
        /// </summary>
        /// <param name="tex">The texture of the car</param>
        /// <returns>The center vector of the car</returns>
        public static Vector2 find_center(Rectangle rec)
```

```
        {
            Vector2 center;
            center = new Vector2((rec.Width) / 2, (rec.Height) / 2);

            return center;
        }

        /// <summary>
        /// Find the quarter center of the car
        /// </summary>
        /// <param name="tex">The texture of the car</param>
        /// <returns>The quarter vector of the car</returns>
        public static Vector2 find_reva(Rectangle rec)
        {
            Vector2 center;
            center = new Vector2((rec.Width) / 4, (rec.Height) / 2);

            return center;
        }

        /// <summary>
        /// Find the three quarter center of the car
        /// </summary>
        /// <param name="tex">The texture of the car</param>
        /// <returns>The three quarter vector of the car</returns>
        public static Vector2 find_shloshtreva(Rectangle rec)
        {
            Vector2 center;
            center = new Vector2(((rec.Width / 4) * 3), (rec.Height /
2));

            return center;
        }

        /// <summary>
        /// This function calculate the radius of the current circle.
        /// </summary>
        /// <param name="tex">The texture of the car</param>
        /// <param name="scale">The scale of the car</param>
        /// <param name="big">Set if we need the radius of the big
circle or regular circle</param>
        /// <returns></returns>
        public float find_radius(Rectangle rec, Vector2 scale, bool
big)
        {
            float w = (rec.Width) * scale.X;
```

```
            float h = (rec.Height) * scale.Y;

            if (!big)
            {
                float min1 = Math.Min(w - center.X, h - center.Y); //
the smaller between height and width
                float min2 = Math.Min(min1, 0 + center.X);
                float min3 = Math.Min(min2, 0 + center.Y);

                return min3;
            }
            else
            {
                if (w - center.X > h - center.Y)
                {
                    return (w - center.X);
                }
                else
                {
                    return h - center.Y;
                }
            }
        }
    }
}
```

Collision.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using System.Threading;

namespace xxx
{
    class Collision
    {
        public static bool EnemyWasHurt;

        /// <summary>
        /// Checking collision with a creature or with an attack ball
        /// </summary>
        /// <param name="hero"></param>
        /// <param name="StatesIndex1"></param>
        /// <param name="enemy"></param>
        /// <param name="StatesIndex2"></param>
        public static void CheckCollision(Animal hero, int
StatesIndex1, Animal enemy, int StatesIndex2)
        {
            EnemyWasHurt = false;

            try
            {
                if
((TheDict.dic[hero.folder][hero.state].BigCircles[StatesIndex1 %

TheDict.dic[hero.folder][hero.state].rec.Count].radius +

TheDict.dic[enemy.folder][enemy.state].BigCircles[StatesIndex2 %

TheDict.dic[enemy.folder][enemy.state].rec.Count].radius) >
                    (hero.Pos - enemy.Pos).Length())
                {
```

```
                    foreach (Circle cir1 in
TheDict.dic[hero.folder][hero.state].AllSmallCircles[StatesIndex1 %

TheDict.dic[hero.folder][hero.state].rec.Count])
                    {
                        foreach (Circle cir2 in
TheDict.dic[enemy.folder][enemy.state].AllSmallCircles[StatesIndex2 %

TheDict.dic[enemy.folder][enemy.state].rec.Count])
                        {
                            #region Collision with animals

                            #region Collision with lizard

                            if (enemy.AnimalType == AnimalType.lizard)
                            {
                                if (hero.AnimalType ==
AnimalType.Cub_Simba)
                                {
                                    if (((hero.jumping &&
hero.jumpspeed >= 0) || hero.falling) &&
                                        enemy.Pos.Y - hero.Pos.Y <= 2
* cir2.radius &&
                                        hero.Pos.X >= enemy.Pos.X -
cir2.radius - 10f &&
                                        hero.Pos.X <= enemy.Pos.X +
cir2.radius + 10f)
                                    {
                                        SmallAnimalsDeath(hero, enemy,
cir2);
                                    }
                                    else
                                    {
                                        if (((!hero.jumping &&
hero.Pos.X >= enemy.Pos.X - cir2.radius - 23f &&
                                            hero.Pos.X <= enemy.Pos.X
+ cir2.radius + 23f)))
                                        {
                                            hero.state =
States.GettingHurt;
                                        }
                                        else if (!hero.jumping &&
enemy.state == States.Slash)
                                        {
                                            hero.state =
States.GettingHurt;
```

```
                                        }
                                    }
                                }
                            }

                            #endregion

                            #region Collision with hedgehog

                            if (enemy.AnimalType ==
AnimalType.hedgehog)
                            {
                                if (hero.AnimalType ==
AnimalType.Cub_Simba)
                                {
                                    if (hero.state != States.Rolling
&& Math.Abs((hero.Pos - enemy.Pos).Length()) <= 65f)
                                    {
                                        hero.state =
States.GettingHurt;
                                    }

                                    if (hero.state == States.Rolling
&& Math.Abs((hero.Pos - enemy.Pos).Length()) <= 20f)
                                    {
                                        enemy.jumpspeed = -8f;
                                        enemy.jumping = true;
                                        enemy.state = States.Running;
                                        enemy.effects =
SpriteEffects.FlipVertically;
                                    }

                                    if (enemy.effects ==
SpriteEffects.FlipVertically)
                                    {
                                        if (((hero.jumping &&
hero.jumpspeed >= 0) || hero.falling) &&
                                            enemy.Pos.Y - hero.Pos.Y
<= 2 * cir2.radius &&
                                            hero.Pos.X >= enemy.Pos.X
- cir2.radius - 20f &&
                                            hero.Pos.X <= enemy.Pos.X
+ cir2.radius + 20f)
                                        {
                                            SmallAnimalsDeath(hero,
enemy, cir2);
```

```
                                        }
                                    }
                                }
                            }

                            #endregion

                            #region Collision with beetle

                            if (enemy.AnimalType == AnimalType.beetle)
                            {
                                if (hero.AnimalType ==
AnimalType.Cub_Simba)
                                {
                                    if (((hero.jumping &&
hero.jumpspeed >= 0) || hero.falling) &&
                                        enemy.Pos.Y - hero.Pos.Y <= 2
* cir2.radius &&
                                        hero.Pos.X >= enemy.Pos.X -
cir2.radius - 40f &&
                                        hero.Pos.X <= enemy.Pos.X +
cir2.radius + 40f)
                                    {
                                        SmallAnimalsDeath(hero, enemy,
cir2);
                                    }
                                    else
                                    {
                                        if (!hero.jumping &&
Math.Abs((hero.Pos - enemy.Pos).Length()) <= 60f)
                                        {
                                            hero.state =
States.GettingHurt;
                                        }
                                    }
                                }
                            }

                            #endregion

                            #region Collision with hyena

                            if (enemy.AnimalType == AnimalType.hyena)
                            {
                                if (hero.AnimalType ==
AnimalType.Cub_Simba)
```

```
                                {
                                    if ((hero.jumping &&
hero.jumpspeed >= 0) &&
                                        !enemy.RunningJump &&
!enemy.jumping &&
                                        enemy.state != States.Roar &&
                                        enemy.Pos.Y - hero.Pos.Y <= 2
* cir2.radius &&
                                        hero.Pos.X >= enemy.Pos.X -
cir2.radius - 10f &&
                                        hero.Pos.X <= enemy.Pos.X +
cir2.radius + 10f)
                                    {
                                        hero.Pos = new
Vector2(hero.Pos.X, enemy.Pos.Y - 1.5f * cir2.radius);
                                        hero.jumpspeed = -10f;
                                        hero.jumping = true;
                                        enemy.hp -= 50;
                                        enemy.state = States.Roar;
                                        hero.state = States.Pouncing;

                                        if (enemy.hp == 0)
                                        {
                                            enemy.state =
States.Dying;

Level.Characters.Remove(enemy);
                                            enemy.color =
Color.Transparent;
                                            Game1.UPDATE_EVENT -=
enemy.Update;
                                            Game1.DRAW_EVENT -=
enemy.DrawAnimal;
                                            enemy.IsDead = true;
                                        }
                                    }


                                    if (Math.Abs((hero.Pos -
enemy.Pos).Length()) <= 95f)
                                    {
                                        hero.state =
States.GettingHurt;
                                    }
                                }
                            }
```

```
                    #endregion

                    #endregion

                    #region Simba is getting hurt

                    if (hero.state == States.GettingHurt &&
!hero.blocked)
                    {
                        if (hero.Pos.X > enemy.Pos.X)
                        {
                            hero.Pos += new Vector2(1f, 0);
                        }

                        else if (hero.Pos.X <= enemy.Pos.X)
                        {
                            hero.Pos += new Vector2(-1f, 0);
                        }
                    }

                    #endregion

                    #region Animal death

                    if (enemy.IsDead && hero.AnimalType ==
AnimalType.Cub_Simba)
                    {
                        hero.state = States.Pouncing;

                        if (Level.LevelNumber == 1 &&
                            enemy.AnimalType ==
AnimalType.hyena)
                        {
                            Thread.Sleep(500);
                            Level.InitSecondLevel();
                        }
                    }

                    #endregion

                    #region ניסוי

                    if (hero.AnimalType ==
AnimalType.Adult_Simba)
                    {
```

```
                                if (Math.Abs((hero.Pos -
enemy.Pos).Length()) <= 180f && !EnemyWasHurt)
                                {
                                    if (hero.state == States.Slash ||
                                        hero.state ==
States.DoubleSlash ||
                                        hero.state ==
States.CrouchSlash)
                                    {
                                        enemy.state =
States.GettingSlashed;
                                        EnemyWasHurt = true;
                                    }
                                }

                                if (Math.Abs((hero.Pos -
enemy.Pos).Length()) <= 140f && hero.state != States.ThrowingEnemy)
                                {
                                    hero.state = States.GettingHurt;
                                }
                            }

                            if (EnemyWasHurt)
                            {
                                if (enemy.effects ==
SpriteEffects.None && hero.effects == SpriteEffects.None)
                                {
                                    enemy.Pos += new Vector2(25f, 0);
                                    enemy.effects =
SpriteEffects.FlipHorizontally;
                                }

                                else if (enemy.effects ==
SpriteEffects.FlipHorizontally && hero.effects ==
SpriteEffects.FlipHorizontally)
                                {
                                    enemy.Pos += new Vector2(-25f, 0);
                                    enemy.effects =
SpriteEffects.None;
                                }

                                EnemyWasHurt = false;
                            }

                            #endregion
                        }
```

```
                }
            }
        }
        catch { }


        #region Collision with an Attack Ball

        if (hero.fireball != null || hero.iceball != null)
        {
            if (hero.fireball != null)
            {
                if (((hero.fireball.Pos - enemy.Pos).Length()) <=
90f)
                {
                    enemy.state = States.GettingSlashed;

                    if (hero.effects == SpriteEffects.None &&
                        enemy.effects == SpriteEffects.None)
                    {
                        enemy.effects =
SpriteEffects.FlipHorizontally;
                    }

                    else if (hero.effects ==
SpriteEffects.FlipHorizontally &&
                        enemy.effects ==
SpriteEffects.FlipHorizontally)
                    {
                        enemy.effects = SpriteEffects.None;
                    }

                    Game1.UPDATE_EVENT -= hero.fireball.Update;
                    Game1.DRAW_EVENT -= hero.fireball.Draw;
                    hero.fireball = null;
                }
            }

            if (hero.iceball != null)
            {
                if (((hero.iceball.Pos - enemy.Pos).Length()) <=
90f)
                {
                    enemy.state = States.GettingSlashed;

                    if (hero.effects == SpriteEffects.None &&
```

```
                            enemy.effects == SpriteEffects.None)
                        {
                            enemy.effects =
SpriteEffects.FlipHorizontally;
                        }

                        else if (hero.effects ==
SpriteEffects.FlipHorizontally &&
                            enemy.effects ==
SpriteEffects.FlipHorizontally)
                        {
                            enemy.effects = SpriteEffects.None;
                        }

                        Game1.UPDATE_EVENT -= hero.iceball.Update;
                        Game1.DRAW_EVENT -= hero.iceball.Draw;
                        hero.iceball = null;
                    }
                }
            }

            #endregion
        }

        public static void SmallAnimalsDeath(Animal hero, Animal
enemy, Circle cir)
        {
            hero.Pos = new Vector2(hero.Pos.X, enemy.Pos.Y - 1.5f *
cir.radius);
            hero.jumpspeed = -10f;
            hero.jumping = true;
            Level.Characters.Remove(enemy);
            enemy.color = Color.Transparent;
            Game1.UPDATE_EVENT -= enemy.Update;
            Game1.DRAW_EVENT -= enemy.DrawAnimal;
            enemy.hp = 0;
            enemy.IsDead = true;
        }
    }
}
```

Gam1.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace xxx
{
    public delegate void UpdateDelegate();
    public delegate void DrawDelegate();

    class Game1 : Microsoft.Xna.Framework.Game
    {
        public static UpdateDelegate UPDATE_EVENT;
        public static DrawDelegate DRAW_EVENT;

        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        public OnlineGame onlineGame;
        public static OnlineState Onlinestate;

        public static int ScreenHeight;
        public static int ScreenWidth;

        public static bool exitGame;
        public static bool IsPaused;
        public static bool PressedPaused;

        public static bool Offline = false; // באונליין שאני הנחה מנקודת יוצא,
מחדל כברירת

        Item item;
```

```
        /// <summary>
        /// Game1 constructor
        /// </summary>
        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
            graphics.PreferredBackBufferHeight = 660; // 690 <- האופטימלי
זה מלא למסך
            graphics.PreferredBackBufferWidth = 1280;
            //graphics.IsFullScreen = true;
            ScreenHeight = graphics.PreferredBackBufferHeight;
            ScreenWidth = graphics.PreferredBackBufferWidth;
            this.IsMouseVisible = true;
            exitGame = false;
        }

        /// <summary>
        /// Initalizing Game1 objects
        /// </summary>
        protected override void Initialize()
        {
            base.Initialize();
            TheDict.Init();
        }

        /// <summary>
        /// Loading the content to the graphics card from the pipeline
        /// </summary>
        protected override void LoadContent()
        {
            spriteBatch = new SpriteBatch(GraphicsDevice);

            S.Init(graphics, spriteBatch, Content);

            Onlinestate = OnlineState.AskingRole;

            LionSounds.init();
            LionSounds.firstlevelsong.Play();
            LionSounds.firstlevelsongIns.IsLooped = true;
            //LionSounds.fireballsound.Play();
            //LionSounds.fireballsoundIns.IsLooped = true;

            Level.InitFirstLevel();
```

```
            float c = 960f;

            for (int i = 1; i <= 3; i++)
            {
                item = new Item(S.cm.Load<Texture2D>("PowerUp"), new
Vector2(c += 100, 2800f), 2.2f, Color.White);
            }
        }

        protected override void UnloadContent()
        {

        }

        /// <summary>
        /// Updating all the game objects
        /// </summary>
        /// <param name="gameTime"></param>
        protected override void Update(GameTime gameTime)
        {
            if (S.gameState == GameStates.MainMenu || S.gameState ==
GameStates.Online)
            {
                S.AllMenues[S.gameState].Update();

                if (exitGame)
                {
                    Exit();
                }
            }

            if (S.gameState == GameStates.Play)
            {
                if (Keyboard.GetState().IsKeyDown(Keys.P) &&
!PressedPaused)
                {
                    PressedPaused = true;
                }

                if (!Keyboard.GetState().IsKeyDown(Keys.P) &&
PressedPaused)
                {
                    IsPaused = !IsPaused;
                    PressedPaused = false;
                }
```

```
                if (Keyboard.GetState().IsKeyDown(Keys.Escape))
                {
                    Exit();
                }

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

            if (S.gameState == GameStates.Online)
            {
                switch (Onlinestate)
                {
                    case OnlineState.AskingRole:
                        break;

                    case OnlineState.host:
                        Console.WriteLine("Switch host");
                        onlineGame = new
HostOnlineGame(int.Parse(File.ReadAllText("port.txt")));
                        onlineGame.OnConnection += new
OnConnectionHandler(onlineGame_OnConnection);
                        onlineGame.Init();
                        Onlinestate = OnlineState.Connecting;
                        break;

                    case OnlineState.join:
                        Console.WriteLine("Switch join");
                        onlineGame = new
JoinOnlineGame(File.ReadAllText("ip.txt"),
int.Parse(File.ReadAllText("port.txt")));
                        onlineGame.OnConnection += new
OnConnectionHandler(onlineGame_OnConnection);
                        onlineGame.Init();
                        Onlinestate = OnlineState.Connecting;
                        break;

                    case OnlineState.Connecting:
                        break;

                    case OnlineState.Playing:
                        onlineGame.hostChar.Update();
                        onlineGame.joinChar.Update();
```

```
                    break;
                }
            }

            base.Update(gameTime);
        }

        /// <summary>
        /// Drawing all the game objects
        /// </summary>
        /// <param name="gameTime"></param>
        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.Transparent);

            if (S.gameState == GameStates.MainMenu || S.gameState ==
GameStates.Online)
            {
                spriteBatch.Begin();
                S.AllMenues[S.gameState].draw();
                spriteBatch.End();
            }

            else
            {
                spriteBatch.Begin(SpriteSortMode.Deferred,
BlendState.AlphaBlend, null, null, null, null, Level.cam.Mat);

                spriteBatch.Draw(Level.BackGroundImage, new Vector2(0,
0), null, Color.White * 1f, 0f,
                        new Vector2(0, 0), S.MapsScale,
SpriteEffects.None, 1f);

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

                spriteBatch.End();

                if (Keyboard.GetState().IsKeyDown(Keys.Tab))
                {
                    spriteBatch.Begin();
                    Level.minimap.Draw();
                    spriteBatch.End();
```

```
                }
            }

            base.Draw(gameTime);
        }

        /// <summary>
        /// Checking if there was a connection between host and join
players
        /// </summary>
        void onlineGame_OnConnection()
        {
            Console.WriteLine("found a connection !!!");
            Onlinestate = OnlineState.Playing;
            S.gameState = GameStates.Play;
            Offline = true;
        }
    }
}
```

HostOnlineGame.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace xxx
{
    class HostOnlineGame : OnlineGame
    {
        /// <summary>
        /// Initializing the port
        /// </summary>
        /// <param name="port"></param>
        public HostOnlineGame(int port)
        {
            this.port = port;
        }

        /// <summary>
        /// Initializing the host and join players
        /// </summary>
        protected override void InitChars()
        {
            hostChar = new Animal(Folders.Cub_Simba, States.Stand,
S.spb, new Vector2(200, 3170), null,
                Color.White, 0f, new Vector2(0, 0), new Vector2(2.2f),
SpriteEffects.None, 1f, AnimalType.Cub_Simba);
            hostChar.baseKeys = new UserBaseKeys(Keys.Up, Keys.Down,
Keys.Left, Keys.Right, Keys.LeftShift,
                Keys.Space, Keys.LeftControl, Keys.LeftAlt);
            joinChar = new Animal(Folders.Cub_Simba, States.Stand,
S.spb, new Vector2(400, 3170), null,
```

```
                Color.White, 0f, new Vector2(0, 0), new Vector2(2.2f),
SpriteEffects.None, 1f, AnimalType.Cub_Simba);
            joinChar.baseKeys = new NonBaseKeys();
        }

        /// <summary>
        ///
        /// </summary>
        protected override void SocketThread()
        {
            TcpListener listener = new TcpListener(IPAddress.Any,
port);
            listener.Start();
            client = listener.AcceptTcpClient();

            reader = new BinaryReader(client.GetStream());
            writer = new BinaryWriter(client.GetStream());
            Console.WriteLine("before RaiseOnConnectionEvent");
            base.RaiseOnConnectionEvent();
            Console.WriteLine("after RaiseOnConnectionEvent");


            while (true)
            {

                WriteCharacterData(hostChar);
                ReadAndUpdateCharacter(joinChar);

                Thread.Sleep(10);
            }
        }
    }
}
```

JoinOnlineGame.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace xxx
{
    class JoinOnlineGame : OnlineGame
    {
        string hostip;

        /// <summary>
        /// Initializing the host ip  and the port
        /// </summary>
        /// <param name="hostip"></param>
        /// <param name="port"></param>
        public JoinOnlineGame(string hostip, int port)
        {
            this.port = port;
            this.hostip = hostip;
        }


        /// <summary>
        /// Initializing the host and join players
        /// </summary>
        protected override void InitChars()
        {
            hostChar = new Animal(Folders.Cub_Simba, States.Stand,
S.spb, new Vector2(200, 3170), null,
                Color.White, 0f, new Vector2(0, 0), new Vector2(2.2f),
SpriteEffects.None, 1f, AnimalType.Cub_Simba);
```

```
            hostChar.baseKeys = new NonBaseKeys();
            joinChar = new Animal(Folders.Cub_Simba, States.Stand,
S.spb, new Vector2(400, 3170), null,
                Color.White, 0f, new Vector2(0, 0), new Vector2(2.2f),
SpriteEffects.None, 1f, AnimalType.Cub_Simba);
            joinChar.baseKeys = new UserBaseKeys(Keys.W, Keys.S,
Keys.A, Keys.D, Keys.LeftShift,
                Keys.Z, Keys.X, Keys.C);
        }

        /// <summary>
        ///
        /// </summary>
        protected override void SocketThread()
        {
            client = new TcpClient();
            client.Connect(hostip, port);

            reader = new BinaryReader(client.GetStream());
            writer = new BinaryWriter(client.GetStream());

            base.RaiseOnConnectionEvent();

            while (true)
            {
                ReadAndUpdateCharacter(hostChar);
                WriteCharacterData(joinChar);

                Thread.Sleep(10);
            }
        }
    }
}
```

Ifocus.cs:

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

namespace xxx
{
    interface Ifocus
    {
        Vector2 Pos { get; }
        float Rot { get; }
    }
}
```

ImageProcess.cs (Page.cs):

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

namespace xxx
{
    class Page
    {
        public List<Rectangle> rec { get; private set; }
        public List<Vector2> org { get; private set; }
        public List<Vector2> FlippedOrg { get; private set; }
        public Texture2D tex { get; private set; }
        public List<Circle> BigCircles;
        public List<Circle> SmallCircles;
        public List<List<Circle>> AllSmallCircles;
        public bool IsFound;
        public string name;

        /// <summary>
        /// Page's constructor
        /// </summary>
        /// <param name="folder"></param>
        /// <param name="state"></param>
        public Page(Folders folder, States state)
        {
            BigCircles = new List<Circle>();
            AllSmallCircles = new List<List<Circle>>();
            rec = new List<Rectangle>();
            org = new List<Vector2>();
            FlippedOrg = new List<Vector2>();
            name = folder.ToString() + "/" + state.ToString();
            Processing(name);
            makeTrans();
        }
```

```
/// <summary>
/// Defining the slow of each State's frames
/// </summary>
/// <param name="state"></param>
/// <returns></returns>
public static int DefineStatesSlows(States state)
{
    if (state == States.Running)
    {
        return 3;
    }

    if (state == States.Running_Jump)
    {
        return 6;
    }

    if (state == States.Stand)
    {
        return 5;
    }

    if (state == States.Jump)
    {
        return 7;
    }

    if (state == States.Hanging)
    {
        return 4;
    }

    if (state == States.Roar)
    {
        return 4;
    }

    if (state == States.Slash)
    {
        return 3;
    }

    if (state == States.DoubleSlash)
    {
        return 4;
    }
```

```
if (state == States.Climb)
{
    return 7;
}

if (state == States.AfterFalling)
{
    return 5;
}

if (state == States.GettingHurt)
{
    return 5;
}

if (state == States.GettingSlashed)
{
    return 3;
}

if (state == States.CrouchSlash)
{
    return 3;
}

if (state == States.Crouch)
{
    return 3;
}

if (state == States.ThrowingEnemy)
{
    return 4;
}

if (state == States.TossedBySimba)
{
    return 4;
}

if (state == States.Pouncing)
{
    return 5;
}
```

```
            if (state == States.Rolling)
            {
                return 5;
            }

            if (state == States.BeetleBeforeDying)
            {
                return 3;
            }

            if (state == States.Swinging)
            {
                return 3;
            }

            return 0;
        }

        /// <summary>
        /// Processing texture's data
        /// </summary>
        /// <param name="name"></param>
        public void Processing(string name)
        {
            tex = S.cm.Load<Texture2D>(name);
            Color[] col = new Color[tex.Width];
            Color[] check = new Color[tex.Width * tex.Height];
            tex.GetData<Color>(0, new Rectangle(0, tex.Height - 1,
tex.Width, 1), col, 0, tex.Width);
            tex.GetData<Color>(check);

            List<int> pnt = new List<int>(); // השחורות הנקודות כל של מערך
מסוים בסטריפ

            for (int i = 0; i < col.Length; i++)
            {
                if (col[i] == col[0])
                {
                    pnt.Add(i); // שאוכל כדי האלה הנקודות מיקומי את מוסיף
בהמשך בהם להשתמש
                }
            }

            IsFound = false;
```

```
            for (int i = 1; i < pnt.Count; i += 2) // oringins - על עובר
ה כל
            {
                for (int row = 0; row < tex.Height - 2; row++)
                {
                    if (check[pnt[i] + (row * tex.Width)] == col[0])
                    {
                        IsFound = true;
                        org.Add(new Vector2(pnt[i] - pnt[i - 1],
row));
                        FlippedOrg.Add(new Vector2(pnt[i + 1] -
pnt[i], row));

                        break;
                    }
                }

                if (!IsFound)
                {
                    org.Add(new Vector2(pnt[i] - pnt[i - 1],
tex.Height - 1)); // אחד מינוס הייט טקס בלי גם אפשר פה
                    FlippedOrg.Add(new Vector2(pnt[i + 1] - pnt[i],
tex.Height - 1));
                }

                rec.Add(new Rectangle(pnt[i - 1], 0, pnt[i + 1] -
pnt[i - 1], tex.Height - 2));
            }


            for (int i = 0; i < rec.Count; i++)
            {
                BigCircles.Add(new Circle(Circle.find_center(rec[i]),
rec[i], new Vector2(S.MapsScale, S.MapsScale), true));
                SmallCircles = new List<Circle>();
                SmallCircles.Add(new
Circle(Circle.find_center(rec[i]), rec[i], new Vector2(S.MapsScale,
S.MapsScale), false));
                SmallCircles.Add(new Circle(Circle.find_reva(rec[i]),
rec[i], new Vector2(S.MapsScale, S.MapsScale), false));
                SmallCircles.Add(new
Circle(Circle.find_shloshtreva(rec[i]), rec[i], new
Vector2(S.MapsScale, S.MapsScale), false));
                AllSmallCircles.Add(SmallCircles);
            }
        }
```

```
        /// <summary>
        /// Making the animation strips transparent
        /// </summary>
        public void makeTrans()
        {
            Color[] data = new Color[tex.Width * tex.Height];
            tex.GetData<Color>(data);
            Color trans = data[0];
            Color black = Color.Black;
            for (int i = 0; i < data.Length; i++)
            {
                if (data[i] == trans)
                {
                    data[i] = Color.Transparent;
                }

                // באמצע השחורות ין'האוריג נקודות את יראו שלא בשביל הוא הזה התנאי
סימבה של הגוף
                // פלטפורמה על טיפוס בזמן
                if (data[i] == black && i < data.Length - 1)
                {
                    data[i] = data[i - 1];
                }
            }

            tex.SetData<Color>(data);
        }
    }
}
```

Item.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace xxx
{
    class Item
    {
        public Texture2D tex;
        public Vector2 Pos;
        public float scale;
        public Color color;

        public Item(Texture2D tex, Vector2 Pos, float scale, Color
color)
        {
            Game1.UPDATE_EVENT += this.UpdateItem;
            Game1.DRAW_EVENT += this.DrawItem;

            this.tex = tex;
            this.Pos = Pos;
            this.scale = scale;
            this.color = color;
        }

        public void UpdateItem()
        {
            if (color != Color.Transparent && Math.Abs((this.Pos -
Level.hero.Pos).Length()) <= 50f)
            {
                Game1.DRAW_EVENT -= this.DrawItem;
```

```
                color = Color.Transparent;
            }
        }

        public void DrawItem()
        {
            S.spb.Draw(tex, Pos, null, Color.White, 0f, new
Vector2(tex.Height / 2, tex.Width / 2), scale, SpriteEffects.None,
1f);
        }
    }
}
```

Level.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace xxx
{
    static class Level
    {
        #region Variables

        public static int LevelNumber;
        public static Animal hero;
        public static Animal scar;
        public static Animal lizard;
        public static Animal hedgehog;
        public static Animal beetle;
        public static Animal hyena;
        public static List<Animal> Characters;

        public static Texture2D BackGroundImage;
        public static Texture2D LogicBackGround;
        public static MiniMap minimap;
        public static Camera cam;

        public static float offset = 500;

        #endregion

        /// <summary>
        /// Initializing all the first level necessary data
        /// </summary>
```

```
        public static void InitFirstLevel()
        {
            LevelNumber = 1;

            LogicBackGround = S.cm.Load<Texture2D>("LogicFirststage");
            BackGroundImage = S.cm.Load<Texture2D>("Firststage");

            // 500 3170
            // 3270 850
            hero = new Animal(Folders.Cub_Simba, States.Stand, S.spb,
new Vector2(500, 3170), null, Color.White, 0f,
                new Vector2(0, 0), new Vector2(2.2f),
SpriteEffects.None, 1f, AnimalType.Cub_Simba);
            hero.baseKeys = new UserBaseKeys(Keys.Up, Keys.Down,
Keys.Left, Keys.Right, Keys.LeftShift, Keys.Space, Keys.Z, Keys.X);

            cam = new Camera(hero, new Vector2(1f), new Viewport(0, 0,
Game1.ScreenWidth, Game1.ScreenHeight));

            Map.check(LogicBackGround);
            minimap = new MiniMap(BackGroundImage, new
Vector2(S.MapsScale, S.MapsScale));

            Characters = new List<Animal>();

            for (int i = 1; i <= 1; i++)
            {
                //scar = new Animal(Folders.scar, States.Stand, S.spb,
new Vector2(800, 3170), null, Color.White, 0f,
                //    new Vector2(0, 0), new Vector2(2.2f),
SpriteEffects.None, 1f, AnimalType.lion);
                //scar.baseKeys = new BotKeys(scar, AnimalType.lion);
                //Characters.Add(scar);

                lizard = new Animal(Folders.lizard, States.Stand,
S.spb, new Vector2(offset += 200, 3170), null, Color.White, 0f,
                    new Vector2(0, 0), new Vector2(2.2f),
SpriteEffects.None, 1f, AnimalType.lizard);
                lizard.baseKeys = new BotKeys(lizard,
AnimalType.lizard);
                Characters.Add(lizard);

                hedgehog = new Animal(Folders.hedgehog, States.Stand,
S.spb, new Vector2(offset += 200, 3170), null, Color.White, 0f,
                    new Vector2(0, 0), new Vector2(2.2f),
SpriteEffects.None, 1f, AnimalType.hedgehog);
```

```
                hedgehog.baseKeys = new BotKeys(hedgehog,
AnimalType.hedgehog);
                Characters.Add(hedgehog);

                beetle = new Animal(Folders.beetle, States.Stand,
S.spb, new Vector2(offset += 200, 3170), null, Color.White, 0f,
                    new Vector2(0, 0), new Vector2(2.2f),
SpriteEffects.None, 1f, AnimalType.beetle);
                beetle.baseKeys = new BotKeys(beetle,
AnimalType.beetle);
                Characters.Add(beetle);

                hyena = new Animal(Folders.hyena, States.Stand, S.spb,
new Vector2(3900, 626), null, Color.White, 0f,
                    new Vector2(0, 0), new Vector2(2f),
SpriteEffects.None, 1f, AnimalType.hyena);
                hyena.baseKeys = new BotKeys(hyena, AnimalType.hyena);
                Characters.Add(hyena);

            }
        }

        /// <summary>
        /// Initializing all the first level necessary data
        /// </summary>
        public static void InitSecondLevel()
        {
            #region Deleting the leftovers of the first level

            if (hero != null)
            {
                Game1.UPDATE_EVENT -= hero.Update;
                Game1.DRAW_EVENT -= hero.DrawAnimal;
            }

            if (Characters != null)
            {
                for (int i = 0; i < Characters.Count; i++)
                {
                    Game1.UPDATE_EVENT -= Characters[i].Update;
                    Game1.DRAW_EVENT -= Characters[i].DrawObject;

                }
            }

            #endregion
```

```
            LevelNumber = 2;

            LogicBackGround =
S.cm.Load<Texture2D>("LogicSecondstage");
            BackGroundImage = S.cm.Load<Texture2D>("Secondstage");

            hero = new Animal(Folders.Adult_Simba, States.Jump, S.spb,
new Vector2(3550, 3800), null, Color.White, 0f,
                new Vector2(0, 0), new Vector2(2.4f),
SpriteEffects.None, 1f, AnimalType.Adult_Simba);
            hero.baseKeys = new UserBaseKeys(Keys.Up, Keys.Down,
Keys.Left, Keys.Right, Keys.LeftShift, Keys.Space, Keys.Z, Keys.X);

            hero.jumpspeed = -16f;
            hero.jumping = true;

            cam = new Camera(hero, new Vector2(1f), new Viewport(0, 0,
Game1.ScreenWidth, Game1.ScreenHeight));

            Map.check(LogicBackGround);
            minimap = new MiniMap(BackGroundImage, new
Vector2(S.MapsScale, S.MapsScale));

            Characters = new List<Animal>();

            scar = new Animal(Folders.scar, States.Stand, S.spb, new
Vector2(3000, 3800), null, Color.White, 0f,
                    new Vector2(0, 0), new Vector2(2.5f),
SpriteEffects.None, 1f, AnimalType.lion);
            scar.baseKeys = new BotKeys(scar, AnimalType.lion);
            Characters.Add(scar);
        }
    }
}
```

Item.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace xxx
{
    class Item
    {
        public Texture2D tex;
        public Vector2 Pos;
        public float scale;
        public Color color;

        public Item(Texture2D tex, Vector2 Pos, float scale, Color
color)
        {
            Game1.UPDATE_EVENT += this.UpdateItem;
            Game1.DRAW_EVENT += this.DrawItem;

            this.tex = tex;
            this.Pos = Pos;
            this.scale = scale;
            this.color = color;
        }

        public void UpdateItem()
        {
            if (color != Color.Transparent && Math.Abs((this.Pos -
Level.hero.Pos).Length()) <= 50f)
            {
                Game1.DRAW_EVENT -= this.DrawItem;
                color = Color.Transparent;
```

```
            }
        }

        public void DrawItem()
        {
            S.spb.Draw(tex, Pos, null, Color.White, 0f, new
Vector2(tex.Height / 2, tex.Width / 2), scale, SpriteEffects.None,
1f);
        }
    }
}
```

LionSounds.cs:

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

namespace xxx
{
    static class LionSounds
    {
        public static SoundEffect firstlevelsong;
        public static SoundEffectInstance firstlevelsongIns;

        public static SoundEffect fireballsound;
        public static SoundEffectInstance fireballsoundIns;

        /// <summary>
        /// Initializing game's audio data
        /// </summary>
        public static void init()
        {
            firstlevelsong =
S.cm.Load<SoundEffect>("Audio/FirstLevelSong");
            firstlevelsongIns = firstlevelsong.CreateInstance();

            fireballsound =
S.cm.Load<SoundEffect>("Audio/FireBallSound");
            fireballsoundIns = fireballsound.CreateInstance();
        }
    }
}
```

Map.cs:

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

namespace xxx
{
    enum GroundType { Platform, Air, Blocked, Can_Hang, Swing }

    static class Map
    {
        static Color[] col;
        public static GroundType[,] Locations;
        public static List<Vector2> HangLocations;
        public static List<Vector2> SwingLocations;
        public static Texture2D tex { get; private set; }

        /// <summary>
        /// Processing the map's texture data
        /// </summary>
        /// <param name="tex"></param>
        public static void check(Texture2D tex)
        {
            Locations = new GroundType[tex.Height, tex.Width];
            HangLocations = new List<Vector2>();
            SwingLocations = new List<Vector2>();
            col = new Color[tex.Width * tex.Height];
            tex.GetData<Color>(col);

            for (int i = 0; i < tex.Height; i++)
            {
                for (int j = 0; j < tex.Width; j++)
                {
                    if (col[tex.Width * i + j] == col[0])
                    {
                        Locations[i, j] = GroundType.Platform;
```

```
                    }
                    else if (col[tex.Width * i + j] == col[1])
                    {
                        Locations[i, j] = GroundType.Blocked;
                    }
                    else if (col[tex.Width * i + j] == col[2])
                    {
                        HangLocations.Add(new Vector2(j, i));
                        Locations[i, j] = GroundType.Can_Hang;
                    }
                    else if (col[tex.Width * i + j] == col[3])
                    {
                        SwingLocations.Add(new Vector2(j, i));
                        Locations[i, j] = GroundType.Swing;
                    }
                    else
                    {
                        Locations[i, j] = GroundType.Air;
                    }
                }
            }
        }
    }
}
```

Menu.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace xxx
{
    class Button
    {
        String text;
        public String name;
        public Vector2 Pos;
        public Color color;

        /// <summary>
        /// Button's constructor
        /// </summary>
        /// <param name="name"></param>
        /// <param name="text"></param>
        /// <param name="pos"></param>
        public Button(String name, String text, Vector2 pos)
        {
            this.name = name;
            this.text = text;
            this.Pos = pos;
            color = Color.Red;
        }

        /// <summary>
        /// Reading the each button's name and changing the the
game/online states respectively
        /// </summary>
```

```
        /// <param name="name"></param>
        public static void ReadButtonName(string name)
        {
            try
            {
                S.gameState =
(GameStates)Enum.Parse(typeof(GameStates), name);

                if (name == "MainMenu")
                {
                    //S.ReloadGame();
                }
            }

            catch
            {
                if (name == "SinglePlayer")
                {
                    S.gameState = GameStates.Play;

                    //Game1.hero = new Lion(Folders.Cub_Simba,
States.Stand, S.spb, new Vector2(200, 3170), null, Color.White, 0f,
                    //    new Vector2(0, 0), new Vector2(2.2f),
SpriteEffects.None, 1f, AnimalType.Cub_Simba);
                    //Game1.hero.baseKeys = new UserBaseKeys(Keys.Up,
Keys.Down, Keys.Left, Keys.Right,
                    //    Keys.LeftShift, Keys.Space, Keys.Z, Keys.X);

                    //Game1.cam = new Camera(Game1.hero, new
Vector2(1f), new Viewport(0, 0, Game1.ScreenWidth,
Game1.ScreenHeight));
                }

                if (name == "exit")
                {
                    Game1.exitGame = true;
                }

                if (name == "Online")
                {
                    S.gameState = GameStates.Online;
                }
            }


            try
```

```
            {
                Game1.Onlinestate =
(OnlineState)Enum.Parse(typeof(OnlineState), name);

                if (name == "host")
                {
                    Console.WriteLine("Pressed Host Button");
                    Game1.Onlinestate = OnlineState.host;
                }

                if (name == "join")
                {
                    Console.WriteLine("Pressed Join Button");
                    Game1.Onlinestate = OnlineState.join;
                }
            }

            catch { }
        }

        /// <summary>
        /// Changing the online game state to "playing"
        /// </summary>
        public static void onlineGame_OnConnection()
        {
            Game1.Onlinestate = OnlineState.Playing;
        }

        /// <summary>
        /// Drawing the button
        /// </summary>
        public void drawButton()
        {
            S.spb.DrawString(S.GameFont, this.text, this.Pos,
this.color);
        }
    }

    class Text
    {
        public string text;
        Vector2 scale;
        Color color;
        Vector2 position;
```

```
        public Text(string text, Vector2 scale, Color color, Vector2
position)
        {
            this.color = color;
            this.text = text;
            this.scale = scale;
            this.position = position;
        }

        public void Draw()
        {
            S.spb.DrawString(S.GameFont, text, position, color, 0f,
Vector2.Zero, scale, SpriteEffects.None, 1);
        }
    }

    class Menu
    {
        public List<Button> buttons = new List<Button>();
        public List<Text> texts = new List<Text>();
        public int select;
        public int numberButtons;
        KeyboardState lastState;
        KeyboardState curState;

        /// <summary>
        /// Updating each menu's data
        /// </summary>
        public void Update()
        {
            #region update selection

            lastState = curState;
            curState = Keyboard.GetState();

            if (curState == lastState && (curState.IsKeyDown(Keys.Up)
|| curState.IsKeyDown(Keys.Down) || curState.IsKeyDown(Keys.Enter)))
                return;

            if (buttons.Count == 0)
                return;

            buttons[select].color = Color.Red;

            if (curState.IsKeyDown(Keys.Up))
            {
```

```
                select--;
            }

            if (curState.IsKeyDown(Keys.Down))
            {
                select++;
            }

            if (select < 0)
            {
                select = numberButtons;
            }

            if (select > numberButtons)
            {
                select = 0;
            }

            #endregion

            #region The Selection

            if (curState.IsKeyUp(Keys.Enter) &&
lastState.IsKeyDown(Keys.Enter))
            {
                Button.ReadButtonName(buttons[select].name);

                if (buttons[select].name == "host" ||
buttons[select].name == "join")
                {
                    buttons = new List<Button>();
                }
            }

            #endregion
        }

        public void draw()
        {
            try
            {
                string menuName = S.gameState.ToString();

                WallPapers sp =
(WallPapers)Enum.Parse(typeof(WallPapers), menuName);
                Texture2D wp = TheDict.WallPapersDic[sp];
```

```
                S.spb.Draw(wp, Vector2.Zero, null, Color.White, 0f,
Vector2.Zero,
                    new Vector2(3.41f, 4.93f), SpriteEffects.None, 1);

            }
            catch { }

            foreach (Button button in buttons)
            {
                buttons[select].color = Color.Blue;
                button.drawButton();

                foreach (Text text in texts)
                {
                    text.Draw();
                }
            }
        }
    }

    class mainMenu : Menu
    {
        /// <summary>
        /// mainMenu's Constructor
        /// </summary>
        public mainMenu()
        {
            buttons = new List<Button>();
            texts.Add(new Text("Lion King", new Vector2(3),
Color.Turquoise, new Vector2(460, 50)));

            buttons.Add(new Button("SinglePlayer", "Start Game", new
Vector2(100, 150)));
            buttons.Add(new Button("Online", "Play Online", new
Vector2(100, 250)));
            buttons.Add(new Button("exit", "Exit", new Vector2(100,
350)));
            numberButtons = buttons.Count - 1;
        }
    }

    class OnlineMenu : Menu
    {
        /// <summary>
        /// OnlineMenu's Constructor
        /// </summary>
```

```
        public OnlineMenu()
        {
            buttons = new List<Button>();
            texts.Add(new Text("Lion King", new Vector2(3),
Color.Turquoise, new Vector2(460, 50)));

            buttons.Add(new Button("host", "Host", new Vector2(100,
150)));
            buttons.Add(new Button("join", "Join", new Vector2(100,
250)));
            buttons.Add(new Button("exit", "Exit", new Vector2(100,
350)));
            numberButtons = buttons.Count - 1;
        }
    }
}
```

MinMap.cs:

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

namespace xxx
{
    class MiniMap
    {
        float zoom = 0.08f;
        SpriteObject sbpo;

        /// <summary>
        /// Minimap's constructor
        /// </summary>
        /// <param name="tex"></param>
        /// <param name="scale"></param>
        public MiniMap(Texture2D tex, Vector2 scale)
        {
            this.sbpo = new SpriteObject(tex, new Vector2(0, 0), null,
Color.White * 1f, 0f,
                Vector2.Zero, new Vector2(scale.X * zoom, scale.Y *
zoom), SpriteEffects.None, 1f);
        }

        /// <summary>
        /// Drawing the minimap
        /// </summary>
        public void Draw()
        {
            this.sbpo.DrawObject();

            SpriteObject.draw_rect(new Vector2(Level.hero.Pos.X *
zoom,
              Level.hero.Pos.Y * zoom - 2), 8, Color.Red);
```

```
            for (int i = 0; i < Level.Characters.Count; i++)
            {
                SpriteObject.draw_rect(new
Vector2(Level.Characters[i].Pos.X * zoom,
                    Level.Characters[i].Pos.Y * zoom - 2), 8,
Color.Blue);
            }
        }
    }
}
```

NonBaseKeys.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace xxx
{
    class NonBaseKeys : BaseKeys
    {
        // NonBaseKeys's constructor
        // It's empty in order to be able create an instance of this
class
        public NonBaseKeys()
        {

        }

        /// <summary>
        /// Returns the UpKey state
        /// </summary>
        /// <returns></returns>
        public override bool UpKey()
        {
            return false;
        }

        /// <summary>
        /// Returns the DownKey state
        /// </summary>
        /// <returns></returns>
        public override bool DownKey()
        {
```

```
        return false;
    }

    /// <summary>
    /// Returns the LeftKey state
    /// </summary>
    /// <returns></returns>
    public override bool LeftKey()
    {
        return false;
    }

    /// <summary>
    /// Returns the RightKey state
    /// </summary>
    /// <returns></returns>
    public override bool RightKey()
    {
        return false;
    }

    /// <summary>
    /// Returns the ShiftKey state
    /// </summary>
    /// <returns></returns>
    public override bool ShiftKey()
    {
        return false;
    }

    /// <summary>
    /// Returns the PunchKey state
    /// </summary>
    /// <returns></returns>
    public override bool PunchKey()
    {
        return false;
    }

    /// <summary>
    /// Returns the FireKey state
    /// </summary>
    /// <returns></returns>
    public override bool FireBallKey()
    {
        return false;
```

```
        }

        /// <summary>
        /// Returns the IceKey state
        /// </summary>
        /// <returns></returns>
        public override bool IceBallKey()
        {
            return false;
        }
    }
}
```

OnlineGame.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace xxx
{
    public delegate void OnConnectionHandler();

    abstract class OnlineGame
    {
        protected BinaryReader reader;
        protected BinaryWriter writer;

        protected Thread thread;

        protected TcpClient client;

        protected int port;

        public Animal hostChar, joinChar;

        public event OnConnectionHandler OnConnection;

        /// <summary>
        ///
        /// </summary>
        protected void RaiseOnConnectionEvent()
        {
            if (OnConnection != null)
            {
                OnConnection();
```

```
        }
    }

    /// <summary>
    ///
    /// </summary>
    public void Init()
    {
        InitChars();
        StartCommunication();
    }

    protected abstract void InitChars();

    /// <summary>
    ///
    /// </summary>
    public void StartCommunication()
    {
        thread = new Thread(new ThreadStart(SocketThread));
        thread.IsBackground = true;
        thread.Start();
    }

    /// <summary>
    ///
    /// </summary>
    /// <param name="player"></param>
    protected void ReadAndUpdateCharacter(Animal player)
    {
        Vector2 temp=new Vector2();
        SpriteEffects effect;
        States state;
        Console.WriteLine("starting reading!!!");
        Console.WriteLine("reading X position");
        temp.X = reader.ReadSingle();
        Console.WriteLine("X:" + temp.X);
        Console.WriteLine("reading Y position");
        Console.WriteLine("Y: "+ temp.Y);
        temp.Y = reader.ReadSingle();
        Console.WriteLine("reading Rot");
        player.Rot = reader.ReadSingle();
        Console.WriteLine("reading effect");
        effect = (SpriteEffects)reader.ReadInt32();
        Console.WriteLine("reading state");
        state = (States)reader.ReadInt32();
```

```
            player.Pos = temp;
            player.effects = effect;
            player.state = state;
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="player"></param>
        protected void WriteCharacterData(Animal player)
        {
            Console.WriteLine("writing X pos");
            writer.Write(player.Pos.X);
            Console.WriteLine("writing Y pos");
            writer.Write(player.Pos.Y);
            Console.WriteLine("writing rot");
            writer.Write(player.Rot);
            Console.WriteLine("writing effects");
            writer.Write((int)player.effects);
            Console.WriteLine("writing state");
            writer.Write((int)player.state);
        }

        protected abstract void SocketThread();
    }
}
```

Program.cs:

```
using System;

namespace xxx
{
#if WINDOWS || XBOX
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        static void Main(string[] args)
        {
            using (Game1 game = new Game1())
            {
                game.Run();
            }
        }
    }
#endif
}
```

SpriteObject.cs:

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

namespace xxx
{
    class SpriteObject : Ifocus
    {
        #region Data

        public Texture2D texture { get; set; }
        public Vector2 Pos { get; set; }
        public Rectangle? sourceRectangle;
        public Color color;
        public float Rot { get; set; }
        public Vector2 origin { get; set; }
        public Vector2 scale;
        public SpriteEffects effects;
        public float layerDepth;

        #endregion

        #region Ctor

        /// <summary>
        /// Initializing the parameters
        /// </summary>
        /// <param name="texture"></param>
        /// <param name="position"></param>
        /// <param name="sourceRectangle"></param>
        /// <param name="color"></param>
        /// <param name="rotation"></param>
        /// <param name="origin"></param>
        /// <param name="scale"></param>
        /// <param name="effects"></param>
```

```
        /// <param name="layerDepth"></param>
        public SpriteObject(Texture2D texture, Vector2 position,
Rectangle? sourceRectangle, Color color, float rotation, Vector2
origin, Vector2 scale, SpriteEffects effects, float layerDepth)
        {
            this.texture = texture;
            this.Pos = position;
            this.sourceRectangle = sourceRectangle;
            this.color = color;
            this.Rot = rotation;
            this.origin = origin;
            this.scale = scale;
            this.effects = effects;
            this.layerDepth = layerDepth;
        }

        #endregion

        #region public funcs

        /// <summary>
        /// Drawing the texture
        /// </summary>
        public virtual void DrawObject()
        {
            S.spb.Draw(texture, Pos, sourceRectangle, color, Rot,
origin, scale, effects, layerDepth);
        }

        /// <summary>
        /// Drawing the ractangle that representing each character in
the minimap
        /// </summary>
        /// <param name="p"></param>
        /// <param name="size"></param>
        /// <param name="color"></param>
        public static void draw_rect(Vector2 p, int size, Color color)
        {
            Texture2D pointTex = new Texture2D(S.gd, 1, 1);
            pointTex.SetData<Color>(new Color[] { Color.White });
            S.spb.Draw(pointTex, new Rectangle((int)p.X - size / 2,
(int)p.Y - size / 2, size, size), color);
        }

        public static Texture2D createCircleText(int radius)
        {
```

```
            radius *= 2;

            Texture2D texture = new Texture2D(S.gd, radius, radius);
            Color[] colorData = new Color[radius * radius];

            float diam = radius / 2f;
            float diamsq = diam * diam;

            for (int x = 0; x < radius; x++)
            {
                for (int y = 0; y < radius; y++)
                {
                    int index = x * radius + y;
                    Vector2 pos = new Vector2(x - diam, y - diam);
                    if (pos.LengthSquared() <= diamsq)
                    {
                        colorData[index] = Color.Green;
                    }
                    else
                    {
                        colorData[index] = Color.Transparent;
                    }
                }
            }

            texture.SetData(colorData);

            return texture;
        }
        #endregion

    }
}
```

Static.cs:

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

namespace xxx
{
    enum OnlineState
    {
        AskingRole, // host or join
        Connecting,
        Playing,
        host,
        join
    }

    enum GameStates
    {
        Play, Exit, Online, MainMenu
    }


    static class S
    {
        #region DATA
        public static OnlineGame onlineGame;
        //public static OnlineState onlinestate;
        public static ContentManager cm;
        public static SpriteBatch spb;
        public static GraphicsDevice gd;
        public static GraphicsDeviceManager gdm;
        public static GameStates gameState;
        public static SpriteFont GameFont;
        public static Dictionary<GameStates, Menu> AllMenues;
        public static float MapsScale;
```

```
        //public static Texture2D tex;
        //public static Map map;

        #endregion

        #region INIT

        /// <summary>
        /// Initializing the parameters
        /// </summary>
        /// <param name="gdm"></param>
        /// <param name="spb"></param>
        /// <param name="cm"></param>
        public static void Init(GraphicsDeviceManager gdm, SpriteBatch
spb, ContentManager cm)
        {
            S.gdm = gdm;
            S.gd = gdm.GraphicsDevice;
            S.spb = spb;
            S.cm = cm;
            S.gameState = GameStates.MainMenu;
            S.GameFont = S.cm.Load<SpriteFont>("SpriteFont1");
            S.AllMenues = new Dictionary<GameStates, Menu>();
            S.AllMenues.Add(GameStates.MainMenu, new mainMenu());
            S.AllMenues.Add(GameStates.Online, new OnlineMenu());
            S.MapsScale = 2.5f;
        }

        #endregion
    }
}
```

TheDict.cs:

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

namespace xxx
{
    enum States
    {
        Stand, Running, Running_Jump, Roar, Jump, Falling,
AfterFalling,
        Hanging, Swinging, Climb, Slash, Pouncing, Rolling,
DoubleSlash, GettingHurt, GettingSlashed,
        Crouch, CrouchSlash, ThrowingEnemy, TossedBySimba,
BeetleBeforeDying
    }

    enum AnimalType { Cub_Simba, Adult_Simba, lion, lizard, hedgehog,
beetle }

    enum Attacks { fireball, iceball }
    enum Folders { Cub_Simba, Adult_Simba, scar, lizard, hedgehog,
beetle }
    enum WallPapers { MainMenu, Online }

    class TheDict
    {
        #region DATA
        public static Dictionary<Folders, Dictionary<States, Page>>
dic;
        public static Dictionary<Attacks, Texture2D> AttacksDic;
        public static Dictionary<WallPapers, Texture2D> WallPapersDic;
        #endregion

        /// <summary>
        /// Initializing the big dictionaries dictionary
```

```
        /// </summary>
        public static void Init()
        {
            dic = new Dictionary<Folders, Dictionary<States, Page>>();
            AttacksDic = new Dictionary<Attacks, Texture2D>();
            WallPapersDic = new Dictionary<WallPapers, Texture2D>();

            foreach (WallPapers wp in
Enum.GetValues(typeof(WallPapers)))
            {
                WallPapersDic.Add(wp,
S.cm.Load<Texture2D>(wp.ToString()));
            }


            foreach (Attacks attack in
Enum.GetValues(typeof(Attacks)))
            {
                AttacksDic.Add(attack,
S.cm.Load<Texture2D>(attack.ToString()));
            }

            foreach (Folders folder in
Enum.GetValues(typeof(Folders)))
            {
                Dictionary<States, Page> temp = new Dictionary<States,
Page>();

                foreach (States state in
Enum.GetValues(typeof(States)))
                {
                    try
                    {
                        temp.Add(state, new Page(folder, state));
                    }

                    catch { }
                }

                dic.Add(folder, temp);
            }
        }
    }
}
```

UserBaseKeys.cs:

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

namespace xxx
{
    class UserBaseKeys : BaseKeys
    {
        Keys Up;
        Keys Down;
        Keys Left;
        Keys Right;
        Keys Shift;
        Keys Punch;
        Keys FireBall;
        Keys IceBall;

        /// <summary>
        /// Initializing the parameters
        /// </summary>
        /// <param name="UpKey"></param>
        /// <param name="DownKey"></param>
        /// <param name="LeftKey"></param>
        /// <param name="RightKey"></param>
        /// <param name="ShiftKey"></param>
        /// <param name="PunchKey"></param>
        /// <param name="FireBallKey"></param>
        /// <param name="IceBallKey"></param>
        public UserBaseKeys(Keys UpKey, Keys DownKey, Keys LeftKey,
Keys RightKey, Keys ShiftKey, Keys PunchKey,
            Keys FireBallKey, Keys IceBallKey)
        {
            //Game1.UPDATE_EVENT += this.Update;

            this.Up = UpKey;
```

```
        this.Down = DownKey;
        this.Left = LeftKey;
        this.Right = RightKey;
        this.Shift = ShiftKey;
        this.Punch = PunchKey;
        this.FireBall = FireBallKey;
        this.IceBall = IceBallKey;

    }

    /// <summary>
    /// Returns the UpKey's state
    /// </summary>
    /// <returns></returns>
    public override bool UpKey()
    {
        return Keyboard.GetState().IsKeyDown(Up);
    }

    /// <summary>
    /// Returns the DownKey's state
    /// </summary>
    /// <returns></returns>
    public override bool DownKey()
    {
        return Keyboard.GetState().IsKeyDown(Down);
    }

    /// <summary>
    /// Returns the LeftKey's state
    /// </summary>
    /// <returns></returns>
    public override bool LeftKey()
    {
        return Keyboard.GetState().IsKeyDown(Left);
    }

    /// <summary>
    /// Returns the RightKey's state
    /// </summary>
    /// <returns></returns>
    public override bool RightKey()
    {
        return Keyboard.GetState().IsKeyDown(Right);
    }
```

```
        /// <summary>
        /// Returns the ShiftKey's state
        /// </summary>
        /// <returns></returns>
        public override bool ShiftKey()
        {
            return Keyboard.GetState().IsKeyDown(Shift);
        }

        /// <summary>
        /// Returns the PunchKey's state
        /// </summary>
        /// <returns></returns>
        public override bool PunchKey()
        {
            return Keyboard.GetState().IsKeyDown(Punch);
        }

        /// <summary>
        /// Returns the FireBallKey's state
        /// </summary>
        /// <returns></returns>
        public override bool FireBallKey()
        {
            return Keyboard.GetState().IsKeyDown(FireBall);
        }

        /// <summary>
        /// Returns the IceBallKey's state
        /// </summary>
        /// <returns></returns>
        public override bool IceBallKey()
        {
            return Keyboard.GetState().IsKeyDown(IceBall);
        }
    }
}
```

עבודות גדולות אינן מבוצעות על ידי כוח, אלא על ידי התמדה

סמואל ג'ונסון