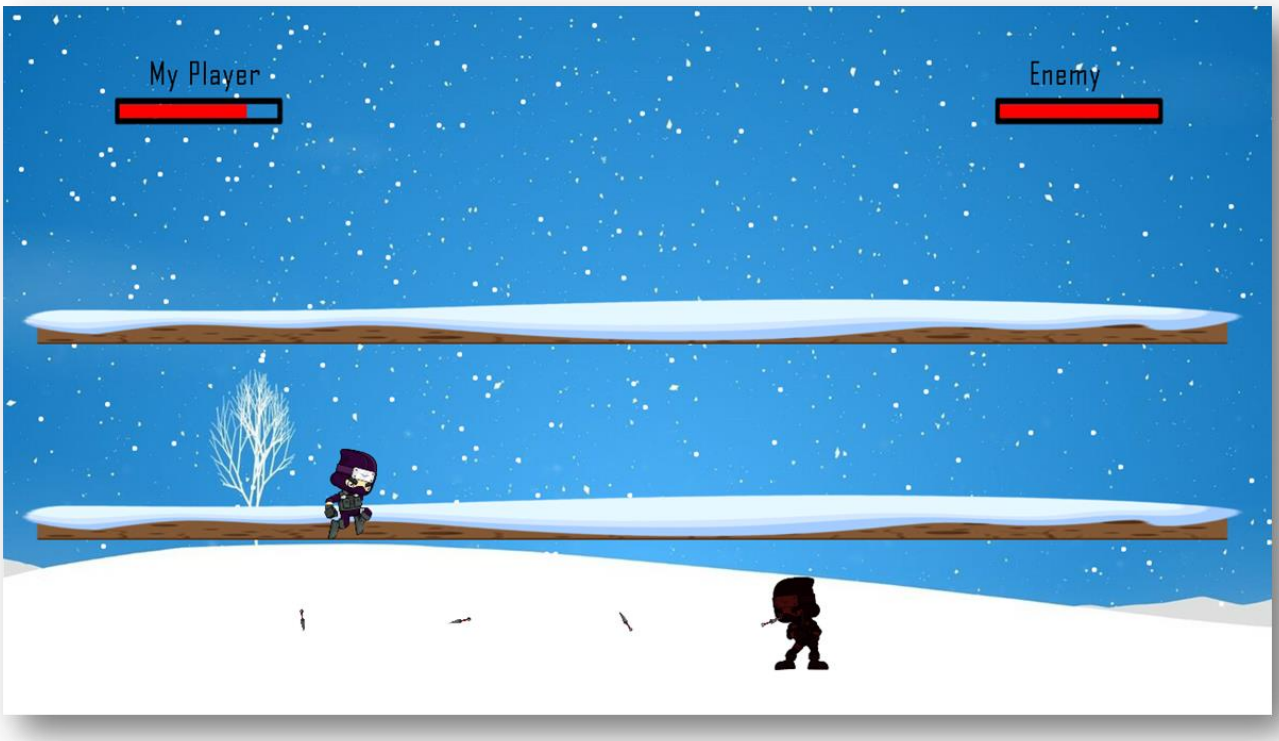
Snow Ninja

משחק אקשן רב משתתפים עם אנימציות המכיל אפשרות למשחק נגד בינה מלאכותית



בית הספר: הכפר הירוק ע"ש לוי אשכול

שם התלמיד: גפן נגה

ת.ז התלמיד: 326670692

שם המנחה: יהודה אור

שם החלופה: סייבר ומערכות הפעלה

תאריך ההגשה: 17.6.2022

תוכן עניינים:

מבוא:

תקציר:

בספר זה אציג לעומק את הפרויקט שכתבתי במהלך שנת הלימוד שלי בכיתה י"ב במסגרת מגמת מדעי המחשב.

הפרויקט שעליו בחרתי לעבוד הוא Snow Ninja- משחק אקשן המכיל אנימציות של שני דמויות נינג'ה שמטרתן להילחם אחת בשנייה על ידי זריקת סכינים. המשחק כולל שני מצבים- מצב שחקן יחיד (Single Player), בו המשתמש יכול לשחק נגד המחשב, ומצב רב-משתתפים (Multiplayer), בו שני משתמשים יכולים לשחק אחד נגד השני על אותו מחשב. בהרצת המשחק, למשתמש יוצג תפריט בו יוכל לבחור לאיזה מצב משחק להיכנס, לקבל הסבר ויזואלי של סיפור הרקע של המשחק והמקשים הזמינים למשחק במקלדת, ולצאת מהמשחק. ישנם כמה אתגרים שאני צופה לי בפרויקט, כגון מימוש האנימציות, זריקת הסכינים וניהול המצבים (States) השונים, אך אני סבור שאני אתגבר על אתגרים אלה בהצלחה.

לאחר שכתבתי ומימשתי את הבסיס של המשחק- האנימציות, הפלטפורמות עליהן הדמויות יכולות לעלות, ניהול זריקת הסכינים ומצבים של ניצחון/הפסד, פיתחתי את אלגוריתם התקשורת והעברת הנתונים בין השרת ללקוח במצב רב המשתתפים, באמצעות שימוש בפרוטוקול TCP. לבסוף, טיפלתי במצב השחקן היחיד- פיתחתי את מנגנון הבינה המלאכותית (בוט) ויצרתי עץ קבלת החלטות הקובע אילו פעולות הבוט יבצע בהתאם לנתונים שהוא מקבל ובהתאם למצב היריב שלו.

הפרויקט שהכנתי פותח בסביבת העבודה Visual Studio בשפת התכנות C# תוך שימוש בספרייה החיצונית Mono, אשר סיפקה כלים נוחים להרצת המשחק ולתצוגה גרפית ברמה גבוהה על המסך.

בחרתי בפרויקט של משחק אקשן זה מכיוון שאני מאוד אוהב משחקי מחשב בכללי ומשחקי אקשן בפרט. בשנים האחרונות ובמיוחד לאור מגיפת הקורונה התחברתי לעולם זה במיוחד וחשבתי שכתיבת פרויקט שכזה תהיה מעניינת ומהנה בשבילי.

ספר פרויקט זה יציג את תהליך העבודה שלי למימוש החלקים השונים בפרויקט וישים דגש על האלגוריתמים המרכזיים בו.

מושגים המוזכרים בעבודה:

- **TCP-** Transmission Control Protocol. פרוטוקול הפועל בתקשורת בשכבת התעבורה לפי מודל חמשת השכבות, ומבטיח העברה אמינה של מידע בין שני מחשבים (שונים או על אותו מחשב באמצעות ניהול תעבורת המידע ב- Threads) בעזרת יצירת חיבור מקושר. לפי פרוטוקול זה, העברת התקשורת מתבצעת באמצעות ארגון המידע ברצף של סיביות הנראה סגמנט. בנוסף לסגמנט, רצף סיביות נוסף עובר בתקשורת ושמו header, והוא אחראי לאימות של איכות התקשורת בין שני גופי התקשורת- פורט התחלתי, פורט יעד והודעת Acknowledge שמאמתת את קבלת ההודעה בין רכיב לרכיב. לאחר שרכיב קיבל את ההודעה, הוא מעביר הודעת Acknowledge שמבטיחה שלא אבד מידע בתקשורת. אם אין אישור הדדי מגופי התקשורת שהתקבלה ההודעה, היא תישלח שוב (The Three Way Handshake).

- **AI/בוט-** אלגוריתם מורכב שמטרתו לדמות התנהגות ודפוסי פעולה של דמות אנושית/שנשלטת בידי אנוש.

- **פולימורפיזם-** תכונה מרכזית בתכנות מונחה עצמים ומבוסס פעולות שלפיה ניתן לרשת תכונות או אלגוריתמים מסוימים מאובייקטים אחרים. המימוש הנפוץ של תכונה זו נעשה במחלקות- ניתן להגדיר מחלקה כמחלקת-אם (Parent), ובכך להוריש תכונות שונות לתת-מחלקות (Child). בכך ניתן ליצור היררכיה של מחלקות- מחלקה יכולה להיות מחלקת-אם לאינספור תת-מחלקות שירשו ממנה תכונות וייעזרו בהן לשימושים שונים. כך ניתן לכתוב קוד אחד (תבנית) ולהשתמש בו למגוון צרכים בצורה פשוטה ללא צורך בכפילויות.

תיאור המשחק:

תחילה, למשתמש יש תפריט ראשוני שמנתב אותו למצבים השונים של המשחק, נותן לו אפשרות לסגור את המשחק ומציע לו מדריך שבו כתוב סיפור הרקע של המשחק והמקשים במקלדת שבאמצעותם משחקים-

A- שמאלה. D- ימינה. S- למטה. E- לזרוק סכין. SPACE- לקפוץ.

בכל אחד מהמצבים המשתמש יוכל להבחין בכך שהיריב כהה יותר, ומטרתו לפגוע ביריבו באמצעות זריקת סכינים ככל האפשר עד שמד החיים שלו יתרוקן.

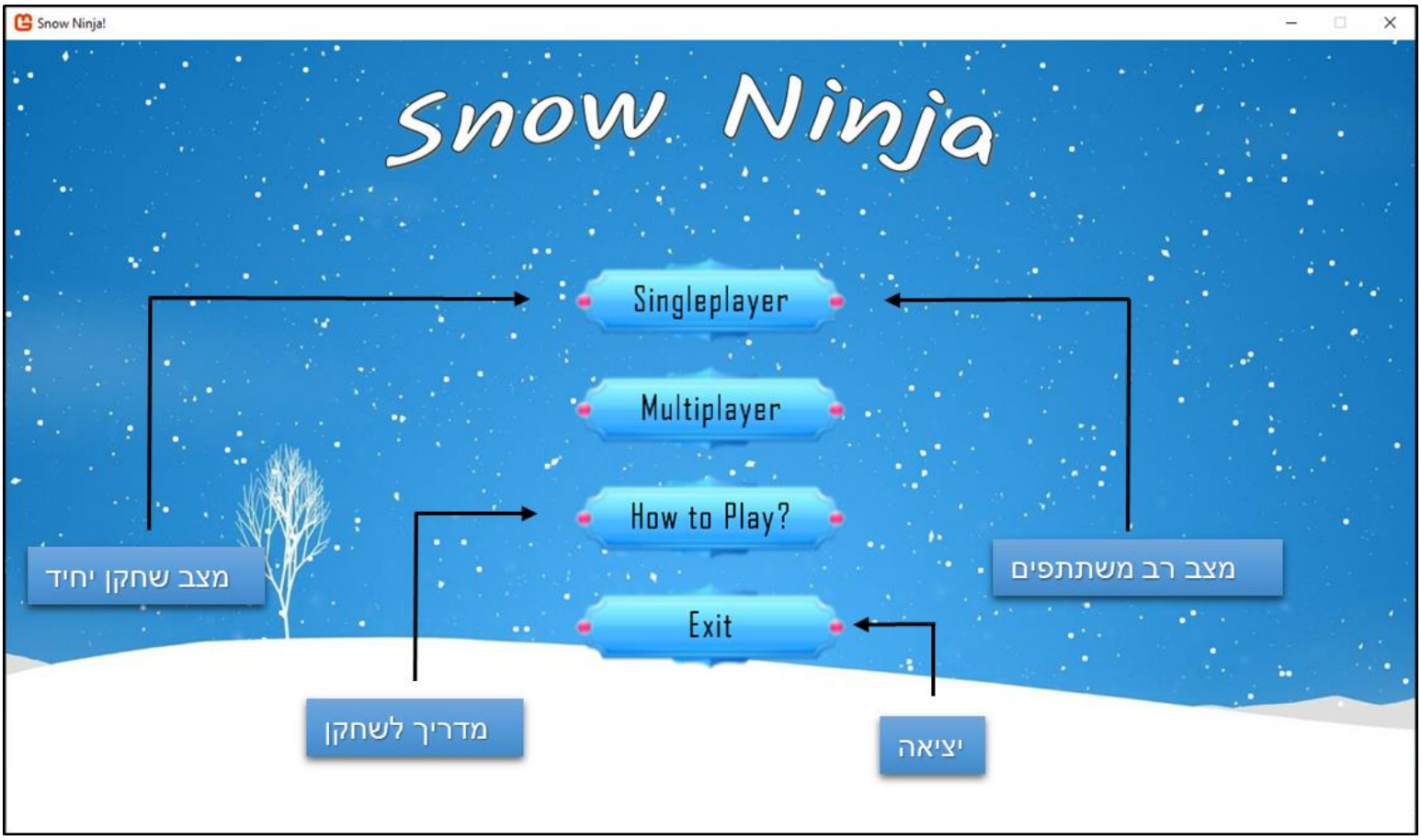
Single Player: שחקן המשחק נגד המחשב- אלגוריתם של בוט שמדמה התנהגות של שחקן אנושי. במקרה של ניצחון, המשחק ייעצר ותופיע תמונת ניצחון על המסך וכפתור שמוציא את המשתמש מהמשחק. במקרה של הפסד, כאשר מד החיים של השחקן מתרוקן, תופיע תמונת הפסד על המסך וכפתור שמוציא את המשתמש מהמשחק.

Multiplayer: שני שחקנים המשחקים על אותו מחשב במסכים שונים. כאשר השחקן הראשון מפסיד, על מסכו תופיע תמונת הפסד ועל מסך השחקן השני תופיע תמונת ניצחון. כאשר השחקן השני מפסיד, על מסכו תופיע תמונת הפסד ועל מסך השחקן הראשון תופיע תמונת ניצחון.

בשני המצבים, במהלך המשחק למשתמש ישנה אפשרות לעצור את המשחק- באמצעות לחיצה על המקש Escape במקלדת, יופיע תפריט של אפשרויות עבור המשתמש- הוא יוכל להמשיך את המשחק כרגיל או לחזור לתפריט הראשי. אם יחזור לתפריט הראשי, המשחק יתאפס ויותחל מהתחלה. במהלך המשחק, לשחקן יש מספר פלטפורמות שבעזרתן יוכל להתחמק מהיריב וליצור אסטרטגיה.

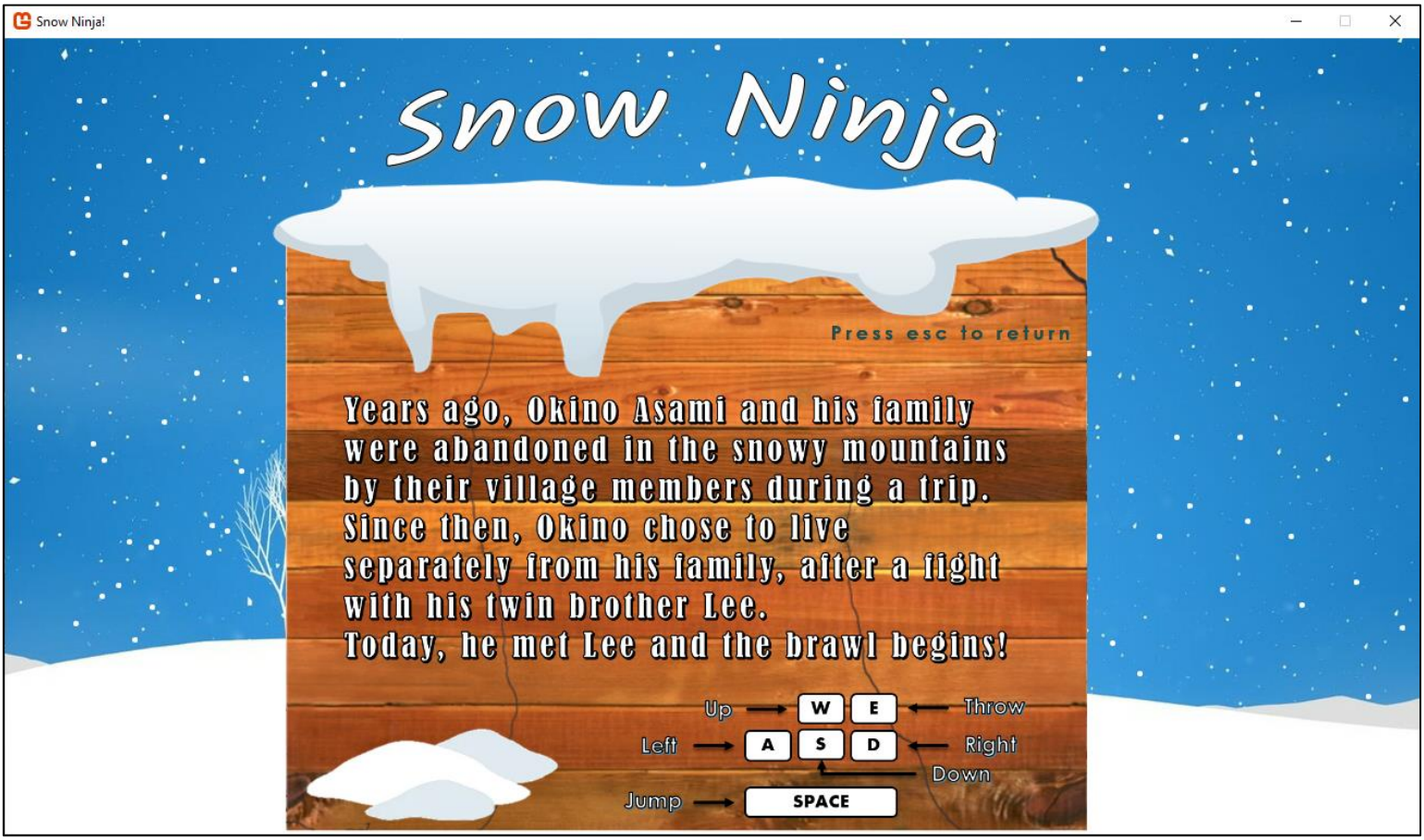
המצבים השונים במשחק:

מצב התחלתי:



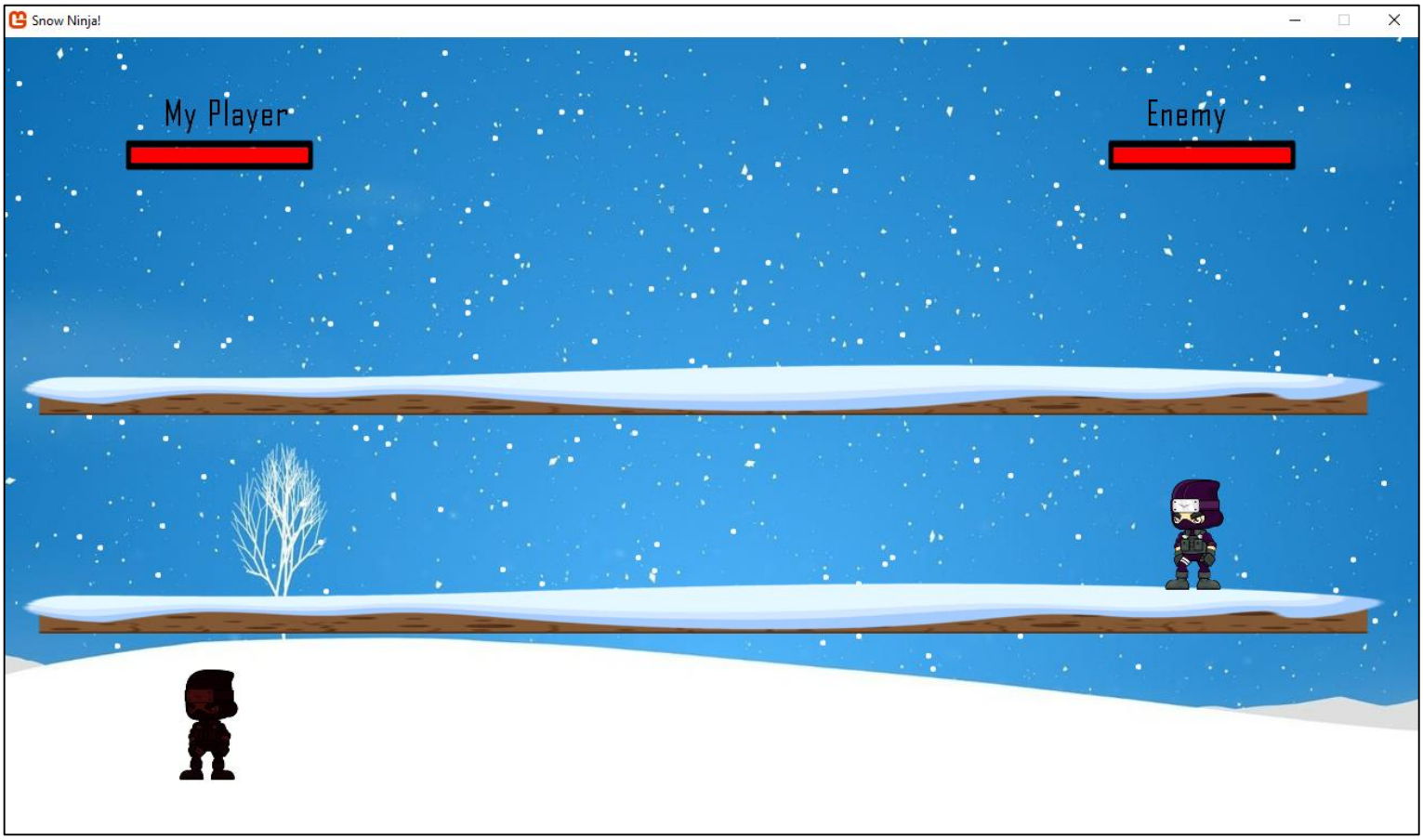
- כאשר פותחים את המשחק, מופיע התפריט הראשי ואנימציה של שלג נופל מאחוריו.
- כל אחד מן הכפתורים מוביל למצב שונה, ויש סאונד של hovering כל פעם שהעכבר עולה על אחד הכפתורים.
- ישנה מוזיקת פתיחה למשחק, הפועלת כאשר המשתמש נמצא בתפריט הראשי.

מדריך למשתמש:



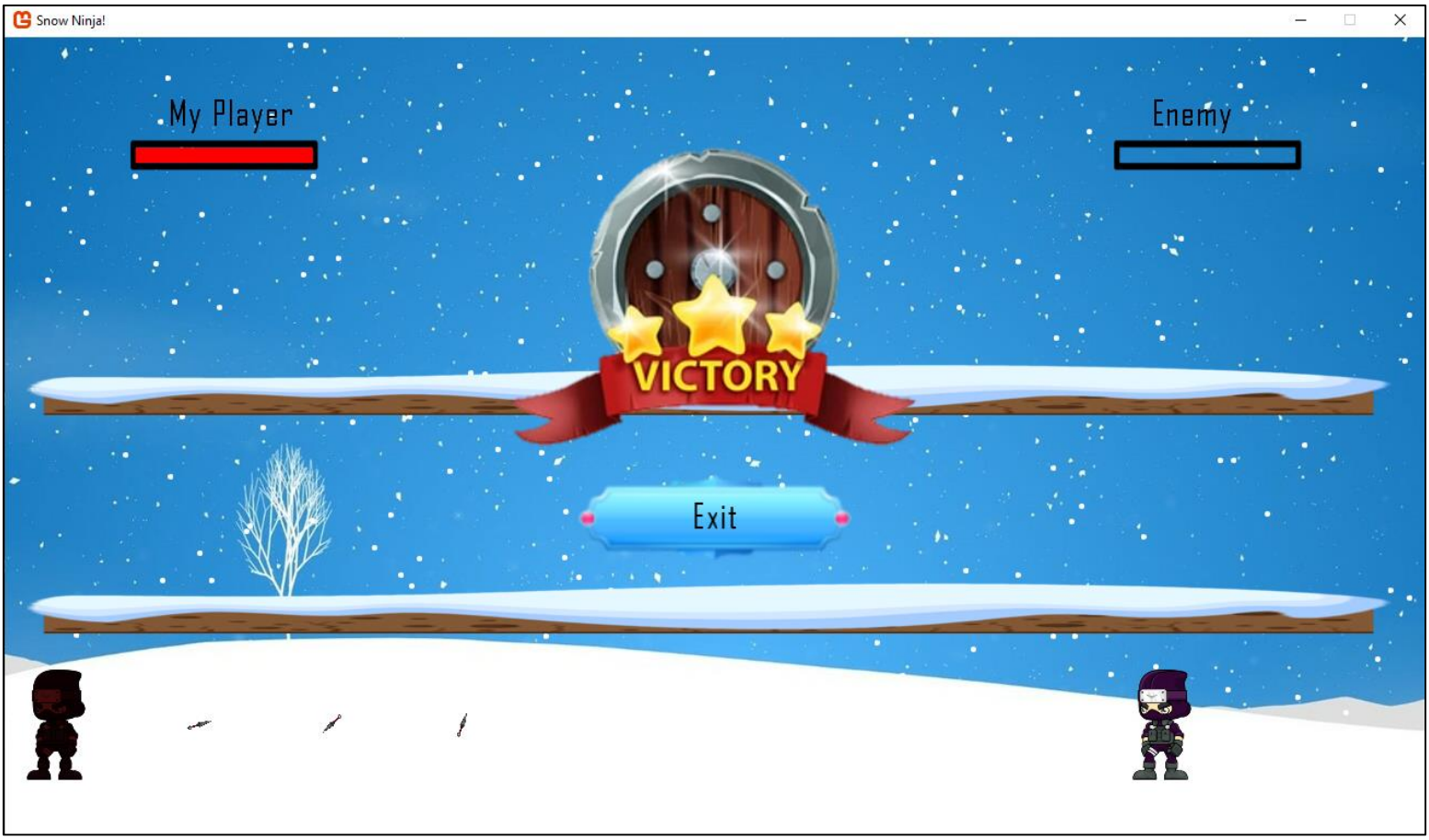
- כאשר המשתמש לוחץ על הכפתור "How to Play?" שבתפריט הראשי, נפתחת תמונה זאת שמכסה את המסך ובה מידע על הרקע של המשחק והמקשים הזמינים למשחק עבור המשתמש.
- באמצעות לחיצה על המקש escape במקלדת, המשתמש יכול לחזור לתפריט הראשי כרגיל.

המשחק עצמו:



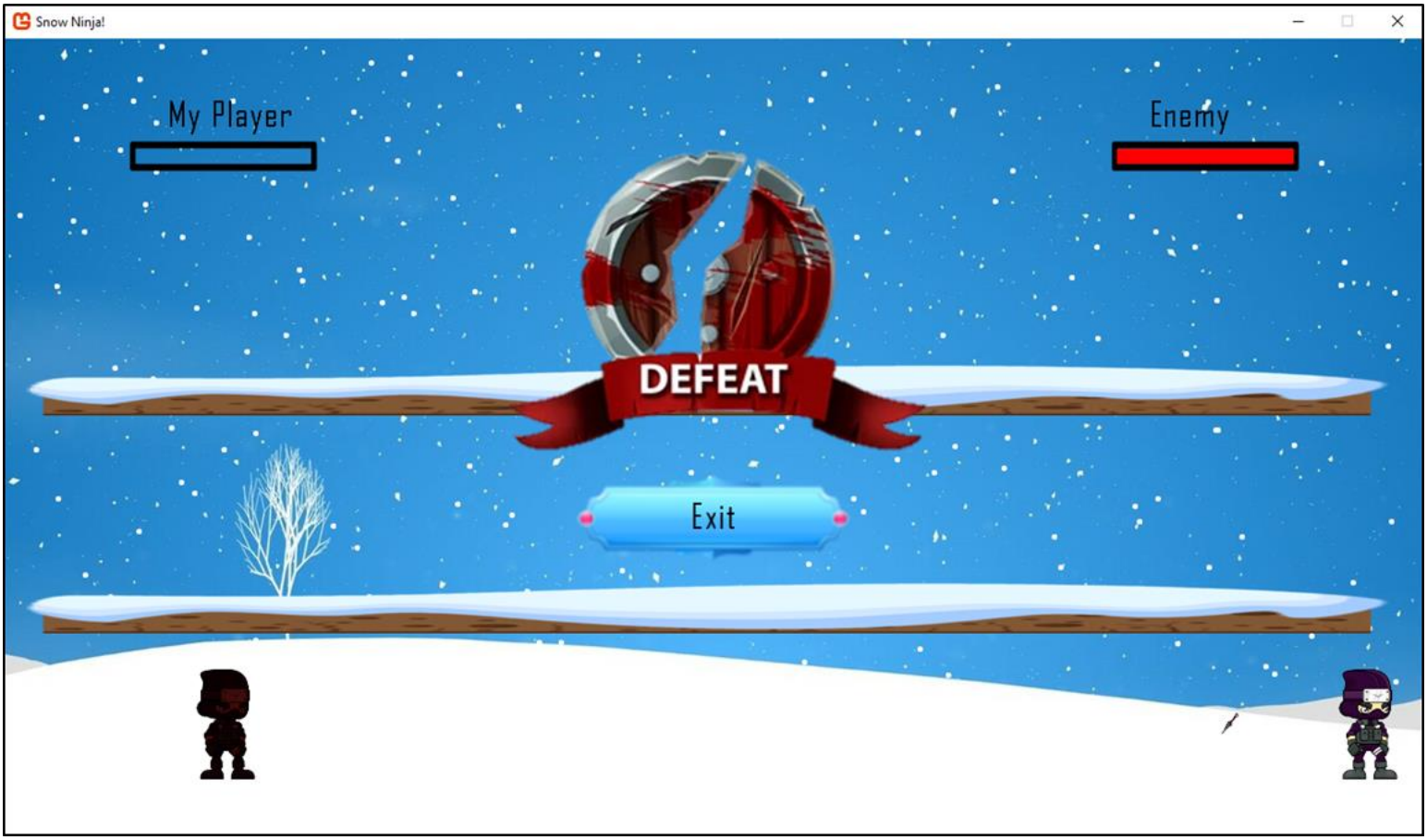
- במצב המשחק (רב משתתפים או שחקן יחיד), השחקן נע במרחב ומולו יריב שצבעו כהה יותר.
- בזירת הקרב של הדמויות נתונות שתי פלטפורמות שעליהם הדמויות יכולות לעלות ולרדת.
- על המסך מצוירים מדי חיים לשחקן וליריב, שמתרוקנים עם כל פגיעה של סכין היריב בדמות.

מצב ניצחון:



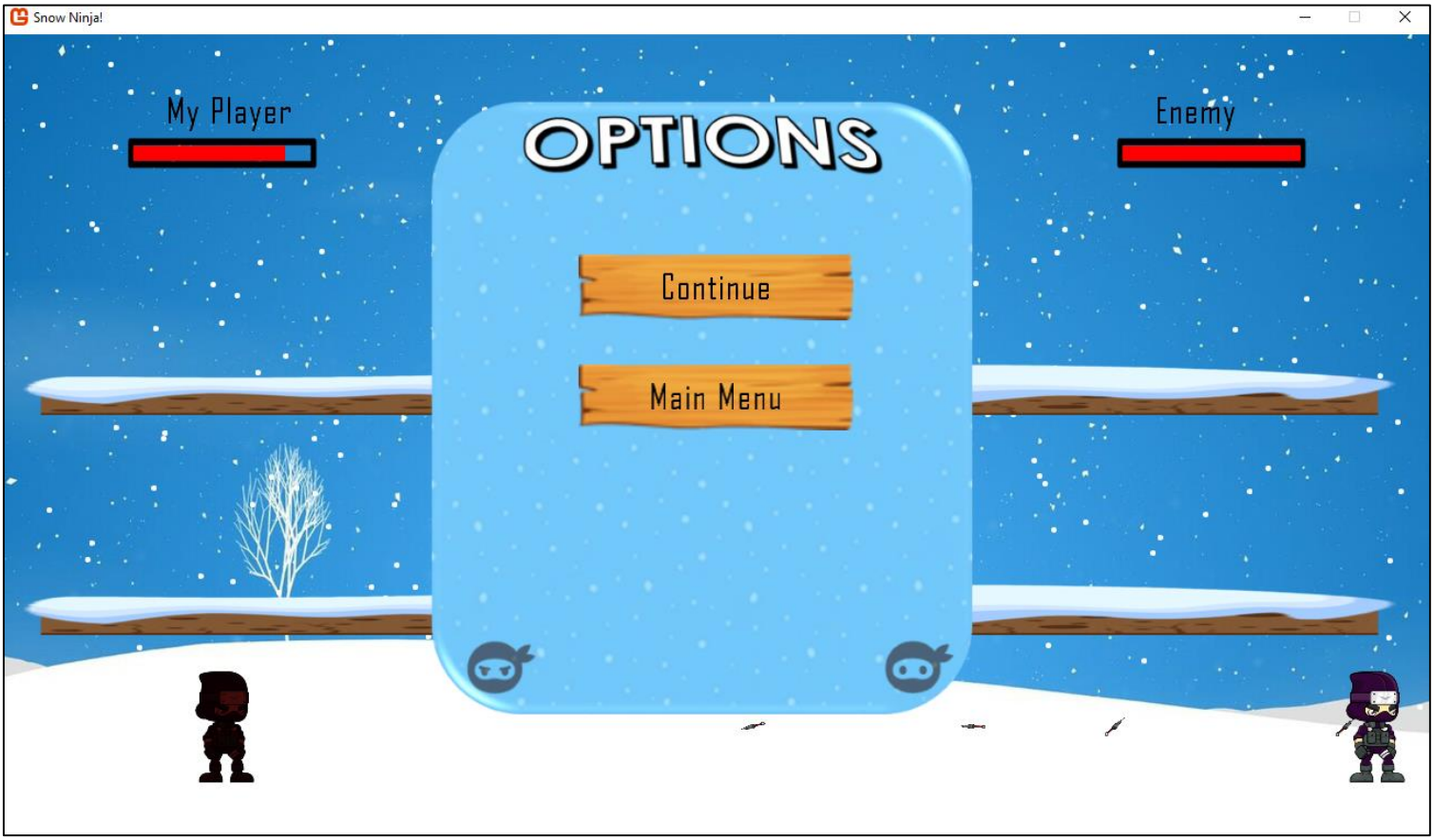
- כאשר מד החיים של היריב מתרוקן ומד החיים של השחקן אינו מרוקן, מופיעה תמונת ניצחון וכפתור ליציאה מהמשחק.
- האובייקטים על המסך קופאים ולא מתעדכנים- המשחק נגמר.

מצב הפסד:



- כאשר מד החיים של השחקן מתרוקן, ומד החיים של היריב אינו מרוקן, מופיעה תמונת הפסד וכפתור ליציאה מהמשחק.
- האובייקטים על המסך קופאים ולא מתעדכנים- המשחק נגמר.

עצירת המשחק:



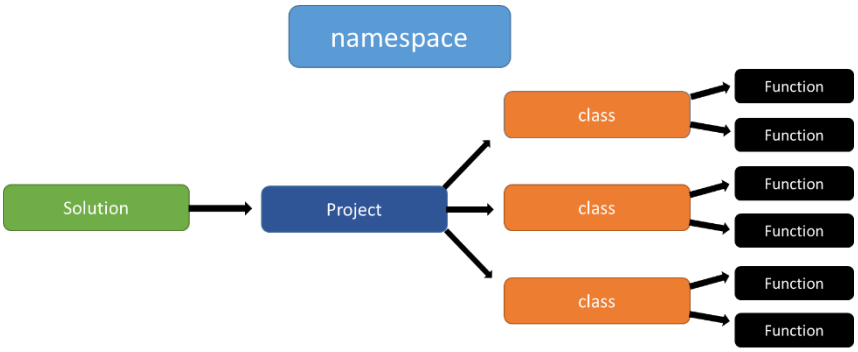
- בלחיצה על המקש escape, מופיע תפריט שכולל שני כפתורים- להמשיך במשחק ולחזור לתפריט הראשי.
- גם במצב זה, המשחק נעצר- האובייקטים אינם מתעדכנים.

חלק תיאורטי:

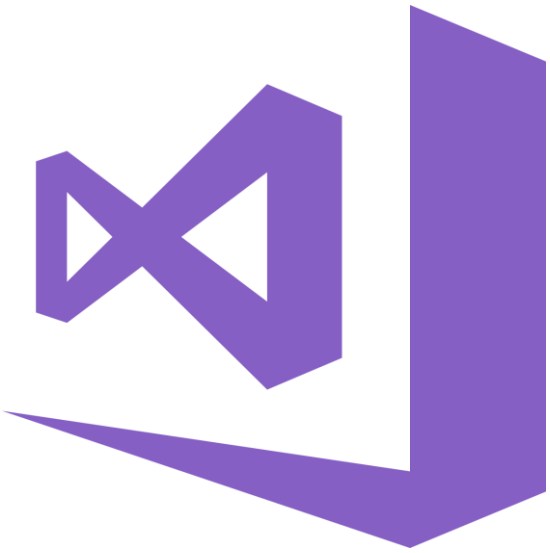
שפת העבודה- C#:

שפת התכנות C# היא שפה עילית אשר פותחה בשנת 2000 על-ידי חברת Microsoft כחלק מפרויקט .NET. שפה זו פותחה על מנת לספק שפת תכנות פשוטה וקלה להבנה ולכתיבה, מודרנית ומונחית עצמים. תחבירה של השפה דומה לזה של שפת התכנות C++ אך כעיקרון השפה מושפעת בעיקר מ- Visual Basic ומ- Java- מהן אומצו מוטיבים כגון תכנות מונחה-אירועים (events, עליהם ארחיב בהמשך), תכנות מונחה- פעולות ועוד.

היררכית הקוד בשפה:



סביבת הפיתוח- Visual Studio:



Visual Studio הינה סביבת פיתוח שנוצרה בידי Microsoft וגרסתה הראשונה יצאה לאור בשנת 1933. סביבה זו מאפשרת למתכנתים לפתח פרויקטים ואתרי אינטרנט דינאמיים. השפות שבהן ניתן לתכנת בסביבה זו הן C++, C#, Visual Basic ועוד.

סביבה זו ידועה בכך שהיא נוחה וידידותית למתכנת ומאפשרת לבצע דיבוג (debug) בצורה יעילה ופשוטה. על מנת להקל על המתכנת, ב- Visual Studio ישנם כלים לפיתוח מהיר וקל- עורך טקסט שצובע מילים בהתאם להקשר שלהם בקוד (משתנה, טיפוס, מילת

מפתח), חלוניות המכילות השלמות אוטומטיות של שמות עצמים ומשתנים שמסייעות לכתיבה מהירה ולמניעת שגיאות ועוד.

הספרייה החיצונית- Mono:

ספריית Mono הינה פרויקט קוד-פתוח שהוקם על ידי החברה Ximian בשנת 2004. ספרייה זו היא כלי שימושי לכתיבה ופיתוח של משחקי מחשב במגוון פלטפורמות.

פלטפורמה זו מיישמת את ממשק XNA 4 של חברת מייקרוסופט שבעזרת הספריות שלו ניתן למתכנתים לעשות שימוש בקוד מובנה בצורה פשוטה ויעילה.

מדוע בחרתי במרכיבים אלה?

- שפת התכנות #C: בחרתי בשפה זו מכיוון שהיא פשוטה להבנה ולכתיבה, מציעה מגוון פעולות שימושיות וקרובה ב- syntax לשפת C ובשפת ++C ששימושיות בעולם התוכנה.
- סביבת הפיתוח Visual Studio: בחרתי בסביבת פיתוח זו משום שהיא נוחה וידידותית למתכנת, מכילה כלים שימושיים עליהם פירטתי לעיל, תומכת בשפת #C וניתן להתקין עליה את הספרייה Mono.
- הספרייה החיצונית Mono: מציעה מחלקות ואובייקטים החיוניים לפיתוח המשחק בגזרת הגרפיקה, התקשורת וההרצה. דוגמה לכך היא האובייקט SpriteBatch, אשר משמש כ"מכחול" על המסך ובעזרתו ניתן להציג תמונות, טקסט ואנימציות על המסך.

שלבים להרצת הפרויקט על המחשב:

1. יש להתקין את סביבת הפיתוח Visual Studio על המחשב- אני התקנתי את הגרסה Visual Studio Community 2017 (Version 15.9), והסבר זה יהיה תקף לגרסה זו.
2. בהגדרות ההתקנה, יש לסמן ב- Workloads את הקופסאות הבאות: .NET desktop development, Desktop Development with C++, Universal Windows Platforms Development, Game development with C++, Visual Studio Extension Development.
3. לאחר ההתקנה של סביבת הפיתוח, יש להתקין את MonoGame 3.7.1 דרך אתר האינטרנט של MonoGame.
4. כעת, סביבת הפיתוח מוכנה להרצת הפרויקט וניהולו.

אובייקטים מובנים בהם אני משתמש בפרויקט:

- GameTime: מספק טיימר ודרך לניהול זמן במשחק.
- SpriteBatch: כלי שדרכו מציגים תמונות וטקסט על המסך.
- MediaPlayer: כלי לניהול ה- Audio של המשחק.
- GraphicsDeviceManager: ניהול התצוגה הגרפית על המסך.
- Song: קטע אודיו ארוך יחסית שמופעל בהתאם לאירועים.
- SoundEffect: קטע אודיו קצר שמופעל בהתאם להתרחשויות במשחק.
- KeyboardState: כלי לניטור מצב המקשים של המקלדת.
- Texture2D: קובץ תמונה די מימדי שניתן להציג על המסך.
- Vector2: כלי המכיל שני ערכים- X ו- Y.
- Rectangle: מחלקה שמכילה פעולות שונות על מלבן.
- Color: כלי שעוזר להגדרת צבעים של אובייקטים במשחק.
- Thread: כלי שמפצל את ריצת הפרויקט לשני אזורים מקבילים של פונקציות.
- ContentManager: כלי לניהול תיקיית content.
- MouseState: כלי לניטור פעילות העכבר.
- SpriteFont: כלי להצגת, עריכת והוספת טקסט על המסך.
- Random: סיפוק מספר רנדומלי.
- MathHelper: סיפוק פעולות מתמטיות רבות לשימוש המפתח.

מבני הנתונים בהם אני משתמש בפרויקט:

- List: רשימה ב-#C היא מבנה נתונים גנרי של רשימה מקושרת שניתן לגשת לאיברים בה לפי מצביע (index), אשר מכיל פעולות מובנות של חיפוש, מיון, הוספה ועוד. הרשימה יכולה להיות מורכבת ממשתנים כגון int, double, char, string ועוד, אך גם מאובייקטים.

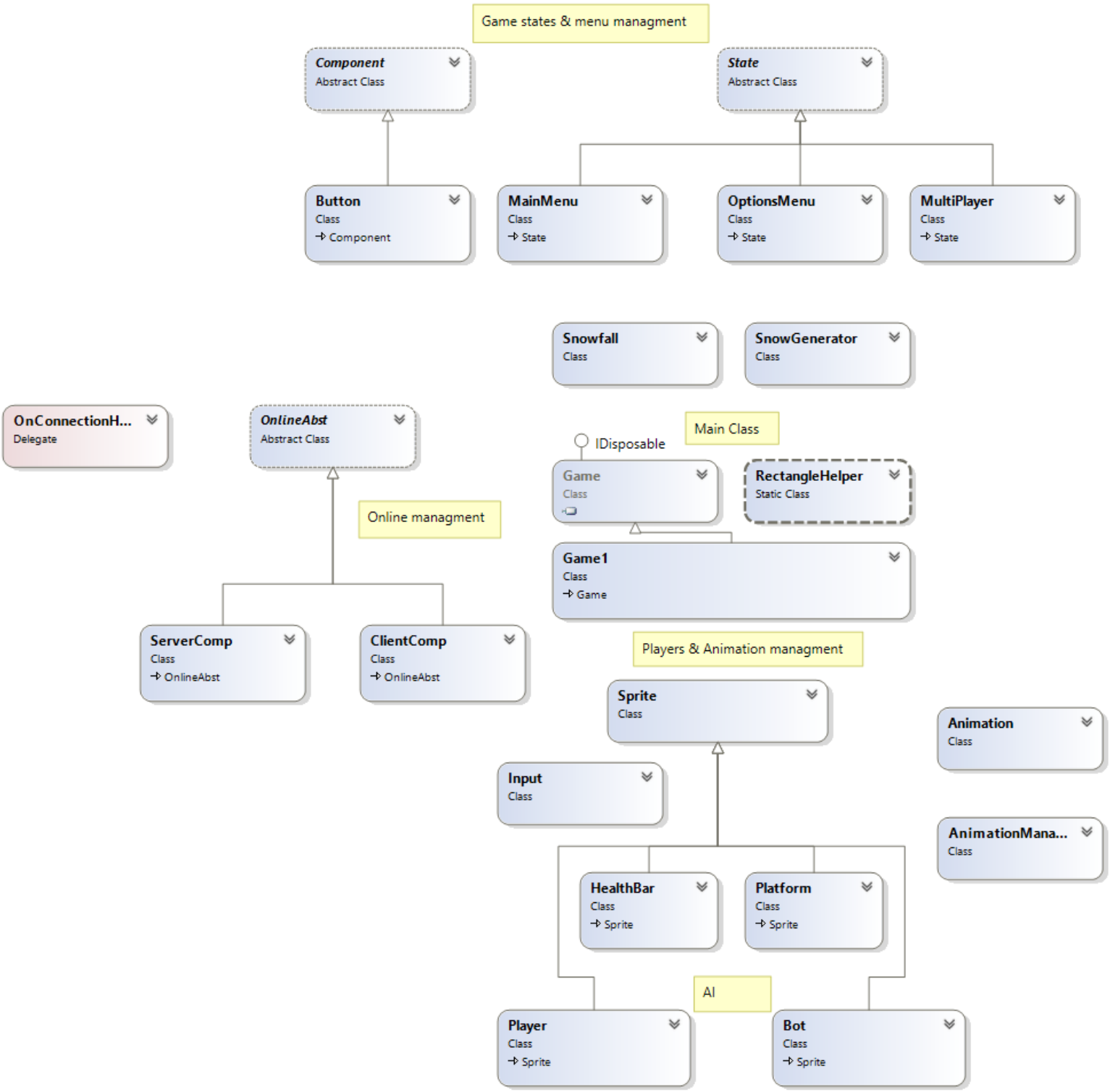
  אם הייתי מממש את המחלקה List, הייתי קודם כל יוצר פעולה בונה שמקצה למבנה הנתונים שהיא מקבלת מקום בזיכרון. לאחר מכן, הייתי יוצר פעולה בונה מעתיקה, שמקבלת List אחר ומעתיקה את ה- List הנתון לאחד חדש. בנוסף, הייתי כותב בשבילה פעולה מאחזרת (Getter) ופעולה קובעת (Setters), ופעולת Add- שמקבלת אובייקט יחיד (מאותו הטיפוס של הרשימה) ומוסיפה אותו לרשימה. עוד מספר פעולות שניתן להוסיף למחלקה כדי לפתחה בנוסף- Clear (מרוקנת את הרשימה), GetLength (מחזירה את אורך הרשימה), Remove (מקבלת אובייקט יחיד, מחפשת אותו ברשימה ומסירה אותו) ו- Sort (ממיינת את הרשימה לפי ערכים מסוימים).

  בפרויקט שלי אני משתמש ברשימה של סכינים, רשימה של פלטפורמות, רשימה של כדורי שלג ורשימה של Components (כפתורים).

- Dictionary: מילון ב- #C הוא מבנה נתונים גנרי המכיל ערכים (Values) בסדר לא מוגדר, שניתן לחפשם באמצעות מפתחות (Keys). מפתחות יכולים להיות משתנים כגון int, string, bool אך גם יכולים להיות אובייקטים, כל עוד הם לא שווים זה לזה ולא שווים null. ערכים יכולים להיות אובייקטים או משתנים ויכולים להיות שווים זה לזה או שווים ל- null. אם הייתי מממש מחלקה Dictionary משלי, תחילה הייתי יוצר פעולה בונה ופעולה בונה מעתיקה, שתקבל ערכים ותעתיק אותם לאובייקט הנתון (this) ולאובייקט שהפעולה מקבלת בכותרת בהתאמה. לאחר מכן הייתי יוצר getters ו- setters, ופעולת Add שמקבלת שני ערכים (לפי הטיפוסים של המילון) ומוסיפה אותם למילון. בנוסף, הייתי כותב פעולת Clear (שמרוקנת את המילון), פעולת GetLength (שמחזירה את כמות הזוגות במילון), פעולת GetValues (שמחזירה רשימה של הערכים במילון), פעולת GetKeys (שמחזירה רשימה של המפתחות במילון), פעולת Remove (שמקבלת מפתח ומסירה את הזוג התואם למפתח) ועוד.

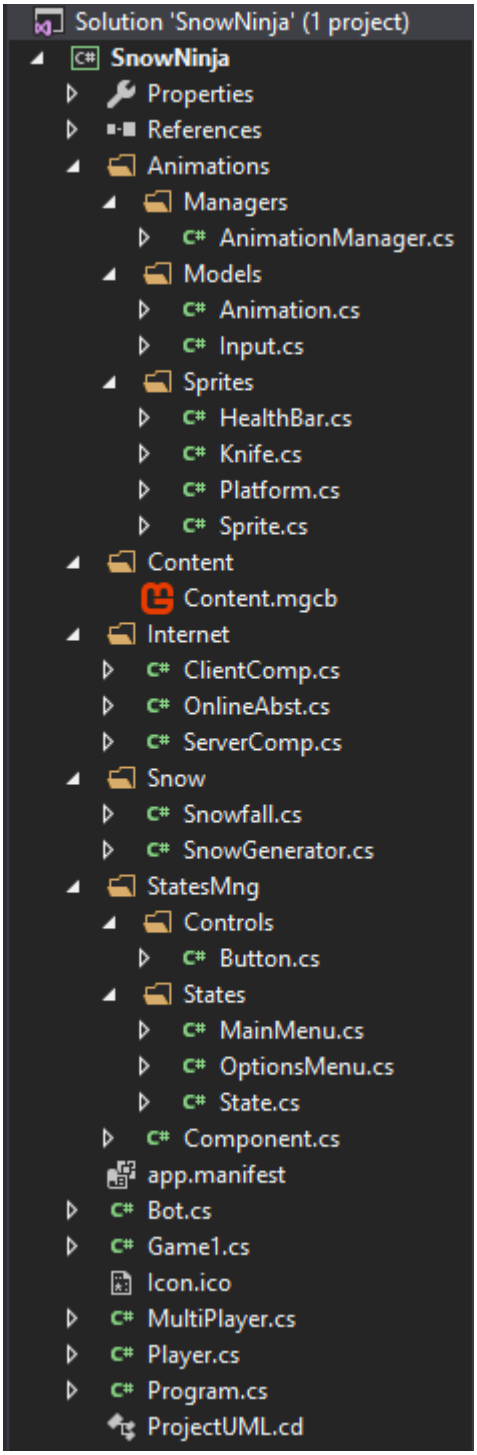
  בפרויקט שלי אני משתמש במילון של אנימציות (הערכים הם קובצי תמונות והמפתחות הם string).

תיאור מחלקות- UML:



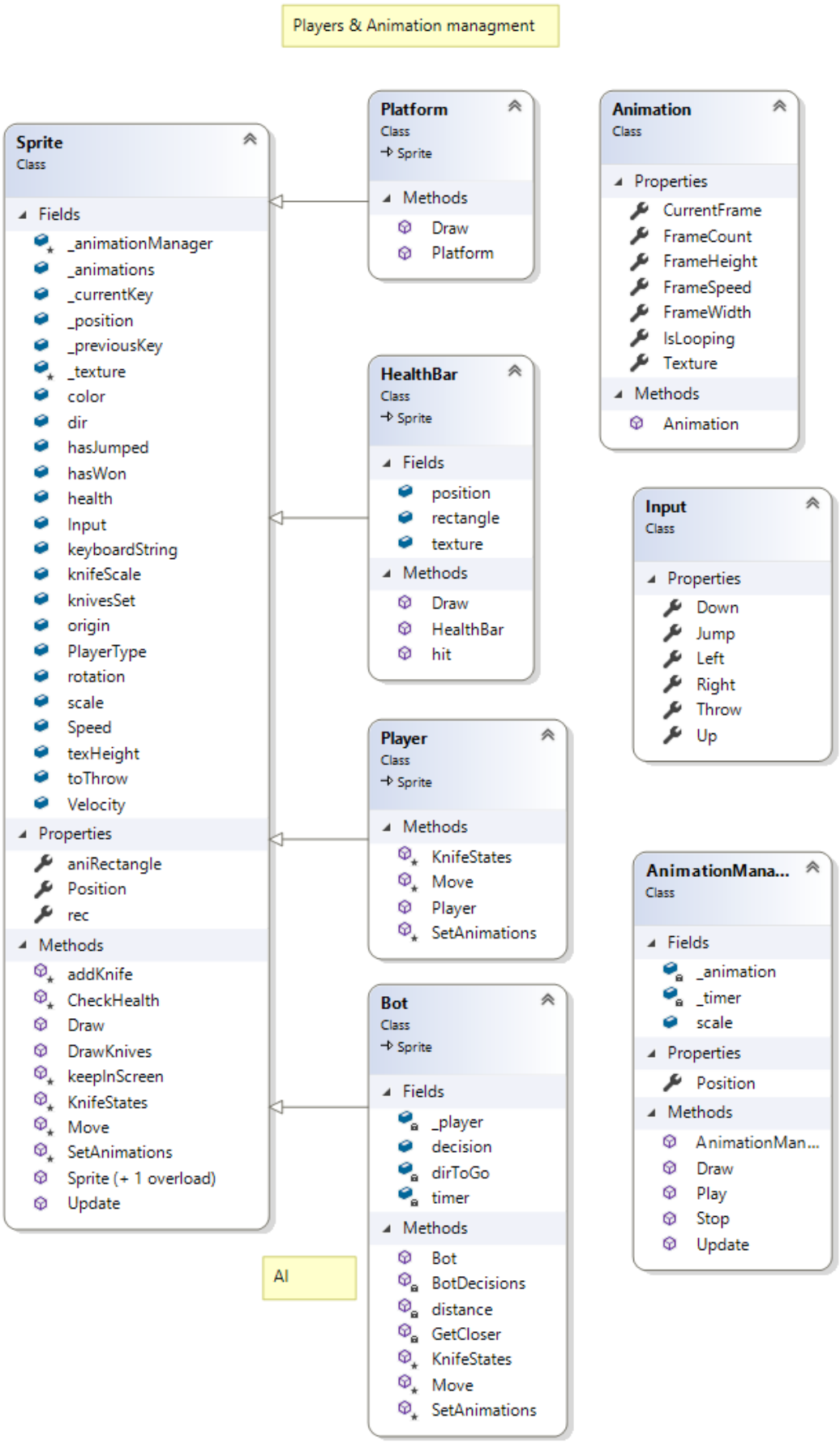
תהליך העבודה ואלגוריתמים מרכזיים:

הקדמה:



- תחילה, מטרתי המרכזית בפרויקט הייתה יצירת משחק שעובד, עבור שני שחקנים בהם שולט משתמש אחד. לאחר מכן הוספתי את הבינה המלאכותית והאינטרנט.
- המטרה הראשונית הייתה אנימציות- יצירת דמות שזזה במרחב וקופצת עם אנימציה חלקה וניהול המשחק- יצירת זריקת הסכינים ופעולת הזריקה, זיהוי פגיעה בשחקן, ניהול מד החיים של השחקן וטיפול במצב של נחיתה על פלטפורמה.
- לאחר מכן, המטרה הייתה ה- States - לנהל פעילות של כפתורים, אירועים (Events), והנראות של מסך הפתיחה. בשלב זה גם הוספתי תמונת רקע, שלג, לוגו ופעילות כפתור המדריך למשתמש. בנוסף לכך, הוספתי את מצב עצירת המשחק- זיהוי לחיצה על מקש escape, עיצוב החלונית והכפתורים ופעילות ה- Events.
- לאחר שסיימתי עם יצירת המשחק עצמו, פיתחתי את הבוט שהמשתמש ישחק נגד במצב ה- Singleplayer- כתבתי אלגוריתם שמשנה את המיקום של הבוט בהתאם למיקום השחקן, הגדרתי באילו תנאים הבוט יזרוק סכין וטיפלתי במצבי קצה.
- בשלב האחרון, עברתי לעבודה על מצב ה- Multiplayer- יצרתי תקשורת בין שרת ללקוח, העברתי נתונים רלוונטיים בין השרת ללקוח באמצעות פרוטוקול TCP, ועבדתי על הנראות של שני המסכים השונים.

חלק ראשון- אנימציות ו- Sprites:



המחלקה Animation:

```
public class Animation
{
    public int CurrentFrame { get; set; }// a field of the index of the current frame in the frames order.

    public int FrameCount { get; set; }// a field of the number of frames the current animation have.

    public int FrameHeight { get { return Texture.Height; } }// a field that contains the height of the frame (identical in every animation)

    public int FrameWidth { get { return Texture.Width / FrameCount; } }// a field that contains thw frame width
                                                                        //(single frame width = entire picture width/number of frames)

    public float FrameSpeed { get; set; }// how fast the frames are going to be drown.

    public bool IsLooping { get; set; }// determines whether the animation will loop or not.

    public Texture2D Texture { get; private set; }// the texture that contains all of the frames in the animation.


    /// <summary>
    /// The constructor of the Animation class.
    /// </summary>

    public Animation(Texture2D texture, int frameCount)
    {
        Texture = texture;

        FrameCount = frameCount;

        IsLooping = true; //sets the default animation as looping.

        FrameSpeed = 0.1f; //sets a custom FrameSpeed of 0.1.
    }
}
```

- המחלקה Animation היא מחלקה פשוטה, ומטרתה להגדיר שדות חיוניים לפעילות האנימציה- אינדקס שמצביע על הפריים הנוכחי שיצויר על המסך, כמות הפריימים של כל אנימציה, הגובה והרוחב של הפריים, מהירות הרצת הפריימים, התמונה שהמחלקה מקבלת ומשתנה בוליאני שקובע האם האנימציה תתבצע בלופ או לא.
- חשוב לציין- על מנת לעזור להבנת הקוד לעיל, הנה דוגמה של תמונה המכילה פריימים:

כל פריים שונה, והרצה שלהם ביחד בעזרת המשתנים לעיל יוצרת את האנימציה.

המחלקה AnimationManager:

```
private Animation _animation;// the current animation that is running.

private float _timer;// helps to manage the animation speed with the variable FrameSpeed

public float scale = 0.35f;// the scale for the Draw function.

public Vector2 Position { get; set; }// position of the animation.

/// <summary>
/// The constructor of the AnimationManager class.
/// </summary>

public AnimationManager(Animation animation)
{
    _animation = animation;
}

/// <summary>
/// draws the current frame using SpriteBatch-
/// cuts from the image the exact frame in order to create the animation.
/// </summary>

public void Draw(SpriteBatch sb, Color PlayerColor)
{
    sb.Draw(_animation.Texture, Position,
        new Rectangle(_animation.CurrentFrame * _animation.FrameWidth,
        0, _animation.FrameWidth, _animation.FrameHeight), PlayerColor, 0,
        new Vector2(_animation.FrameWidth / 2, _animation.FrameHeight / 2), scale, SpriteEffects.None, 1);

}


/// <summary>
/// gets an animation and creates the animation.
/// </summary>
```

- הפעולה הבונה מקצה מקום בזיכרון לאובייקט Animation, כדי שהפעולות בהמשך המחלקה ישתמשו בו.
- הפונקציה Draw מציירת פריים אחד בכל פעם- בעזרת ה- SpriteBatch מצויר על המסך חלק מן התמונה שמכילה את כל הפריימים, כאשר השורה:

```
new Rectangle(_animation.CurrentFrame * _animation.FrameWidth,
    0, _animation.FrameWidth, _animation.FrameHeight);
```

מגדירה איזה חלק ייגזר מהתמונה הגדולה בכל פריים.

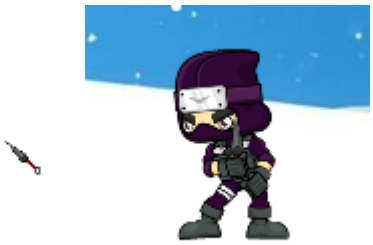
```
public void Play(Animation animation)
{
    if (_animation == animation)
        return;

    _animation = animation;

    _animation.CurrentFrame = 0;

    _timer = 0;
}


/// <summary>
/// a function that stops the animation by reseting the frame index and the timer.
/// </summary>

public void Stop()
{
    _timer = 0;

    _animation.CurrentFrame = 0;
}


/// <summary>
/// creates the frame speed and continues to the next frame
/// every time the timer has reached the max.
/// </summary>

public void Update(GameTime gameTime)
{
    _timer += (float)gameTime.ElapsedGameTime.TotalSeconds;

    if (_timer > _animation.FrameSpeed)
    {
        _timer = 0;

        _animation.CurrentFrame++;

        if (_animation.CurrentFrame >= _animation.FrameCount)
            _animation.CurrentFrame = 0;
    }
}
```

- אפקט האנימציה פועל לפי טיימר והמשתנה FrameSpeed- הטיימר מתעדכן לפי gameTime ואת המשתנה FrameSpeed ניתן להגדיר בצורה ידנית. בפעולה Update, מוגדר שברגע שעברו FrameSpeed שניות, הטיימר יתאפס והאנימציה תעבור לפריים הבא. מטרת התנאי האחרון היא לטפל במצב של סיום האנימציה הנוכחית- אם האינדקס של הפריימים הגיע לכמות הפריימים באנימציה, הוא יתאפס והאנימציה תתחיל מהתחלה.

המחלקה Knife:

```
public class Knife : Sprite
{
    public bool IsRemoved = false; // controls whether to delete the knife or not in PostUpdate in Game1.

    public string knifeDirection; // determines which direction the knife is being thrown to.

    /// <summary>
    /// inherits the constructor from its parent, Sprite.
    /// </summary>

    public Knife(Texture2D texture) : base(texture)
    {

    }

    /// <summary>
    /// creates the flipping motion of the knife while updating its position.
    /// </summary>

    public void UpdateKnife(GameTime gameTime)
    {
        if(knifeDirection == "right")
        {
            Rotation -= 0.3f;
            _position.X += 10f;
        }
        if(knifeDirection == "left")
        {
            Rotation += 0.3f;
            _position.X -= 10f;
        }
    }
}
```

- המחלקה Knife יורשת מהמחלקה Sprite- היא יכולה להשתמש במשתנים שלה, בפעולות האבסטרקטיות שלה ובעוד תכונות. המשתנה הבוליאני IsRemoved חיוני לבדיקת מדי החיים של הדמויות ולמחיקת הסכין מהמסך- פעולות שקורות בפעולה PostUpdate שנמצאת במחלקה Game1.
- ניתן לראות שהפעולה הבונה של Knife ריקה במחלקה זו, אך בזכות הפקודה `: base(texture)`, המחלקה Knife תשתמש בפעולה הבונה של מחלקת האם- Sprite, ותקצה מקום בזיכרון למשתנה texture.
- יצרתי את הפעולה UpdateKnife כדי לעדכן את כיוון וסיבוב הסכין- ברגע שהסכין נזרקת, המשתנה מסוג string בשם knifeDirection שומר בתוכו את כיוון הדמות ברגע הזריקה, וכפי שניתן לראות בקוד- ערך ה-X של הסכין והתנועה הסיבובית של הסכין (Rotation) משתנים בהתאם לכיוון הדמות.



המחלקה HealthBar:

```
public class HealthBar : Sprite
{
    public Texture2D texture;// The bar's texture

    public Vector2 position;// The bar's position on the screen

    public Rectangle rectangle;// The bar's rectangle.

    /// <summary>
    /// The constructor of the HealthBar class
    /// </summary>

    public HealthBar(Texture2D newTex, Vector2 newPos, Rectangle newRec) : base(newTex)
    {
        texture = newTex;
        position = newPos;
        rectangle = newRec;
    }

    /// <summary>
    /// reducing the width of rectangle by 20 pixels.
    /// </summary>

    public void hit()
    {
        rectangle.Width -= 20;
    }

    /// <summary>
    /// draws the health bar on the screen.
    /// </summary>

    public override void Draw(SpriteBatch sb)
    {
        sb.Draw(texture, position, rectangle, Color.White, 0, Vector2.Zero, 0.4f, SpriteEffects.None, 1);
    }
}
```

- מחלקה שמוגדרת כאחת התכונות של המחלקה Sprite (פירוט בהמשך)- שתכונותיה הן טקסטורה, מיקום ומלבן שעוטף את הטקסטורה.
- הפעולה Draw מדפיסה את המד על המסך.
- בפעולה hit, רוחב המלבן קטן ב-20 פיקסלים בכל קריאה לפעולה- מלבן זה עובר לפעולה Draw של המחלקה SpriteBatch בתור sourceRectangle- אשר חותך את הטקסטורה בהתאם למידות המלבן.

המחלקה Sprite:

```
public class Sprite
{
    #region Fields

    protected AnimationManager _animationManager; //managment of animation of the Sprite.

    public Dictionary<string, Animation> _animations;// contains the sets of animation of the Sprite.

    public Vector2 _position;

    protected Texture2D _texture;//in case of using the Sprite class without animations.

    #endregion

    #region Properties

    public Input Input;// objects that contains types of Keys.

    public KeyboardState _currentKey;//Keyboard managment
    public KeyboardState _previousKey;//Keyboard managment

    public string keyboardString = "";// used in the Internet part.

    public float scale;

    public HealthBar health;

    public float Rotation = 0f;

    public bool hasWon = false;// boolean that changes if the sprite has won or not.

    public Vector2 origin;
```

```
    public Vector2 Position // a field that sets the position to the animation if there is an animation.
    {
        get { return _position; }
        set
        {
            _position = value;

            if (_animationManager != null)
                _animationManager.Position = _position;
        }
    }

    public Rectangle Rectangle
    {
        get
        {
            return new Rectangle((int)_position.X, (int)_position.Y,
                (int)(Game1.anyPlayerTex.Width * (10 / scale)), (int)(Game1.anyPlayerTex.Height * scale));
        }
    }

    public float Speed = 3f;

    public Vector2 Velocity;// the speed & direction of the sprite

    public float knifeScale; // modifying the scale of the knife.

    public float texHeight;

    public bool hasJumped;

    public string direction = "none";

    public List<Knife> knivesSet = new List<Knife>();// the knives the sprite is throwing.

    public Color Color;

    public string PlayerType;// whether "Player" or "Enemy".
```

- מחלקה של יצירה, ניהול וציור של אובייקטים על המסך, בנוסף לשימוש באנימציות.
- המשתנים של המחלקה Sprite- חלק מהמשתנים (Origin, Color, Rectangle, Rotation ועוד) נועדו בעיקר בשביל הפעולה Draw, בעוד שמשתנים אחרים (KnivesSet, health, Velocity, animations ועוד) נועדו בשביל פעולות העדכון והתחזוקה של הדמות.

```
/// <summary>
/// Sprite's animations contsructor.
/// </summary>

public Sprite(Dictionary<string, Animation> animations)
{
    _animations = animations;
    _animationManager = new AnimationManager(_animations.First().Value);

    hasJumped = false;

}

/// <summary>
/// Sprite's regular constructor.
/// </summary>
public Sprite(Texture2D texture)
{
    _texture = texture;
    hasJumped = false;

}

/// <summary>
/// part 1: regular Sprite's Draw.
/// part 2: animations's Draw.
/// </summary>
public virtual void Draw(SpriteBatch spriteBatch)
{
    if (_texture != null)
    {
        spriteBatch.Draw(_texture, Position, null, Color, Rotation,
            origin, scale, SpriteEffects.None, 1);
        health.Draw(spriteBatch);
        DrawKnives(spriteBatch);
    }
    else if (_animationManager != null)
    {
        _animationManager.Draw(spriteBatch, Color);
        health.Draw(spriteBatch);
        DrawKnives(spriteBatch);
    }
}

/// <summary>
/// draws the knives.
/// </summary>
public virtual void DrawKnives(SpriteBatch sb)
{
    foreach(var knife in knivesSet)
    {
        sb.Draw(Game1.knifeTexture, knife.Position, null, Color.White, knife.Rotation,
            knife.origin, knifeScale, SpriteEffects.None, 1);
    }

}
```

- למחלקה Sprite יש שתי פעולות בונות- אחת לשם שימוש במחלקה Sprite עבור דמויות דו ממדיות רגילות, והשנייה עבור דמויות עם אנימציה המשתמשות במילון של אנימציות. הפעולה הבונה של AnimationManager מקבלת Animation לכן היא מקבלת את האובייקט הראשון מסוג Animation במילון.
- בדומה לפעולה הבונה, גם פעולת Draw מחולקת לשני מצבים- טיפול במצב של Sprite רגיל וטיפול במצב של אנימציה.

```
/// <summary>
/// updates the position so it stays in screen borders.
/// </summary>

protected void keepInScreen()
{
    _position.X = MathHelper.Clamp(_position.X,
        _animations["runningR"].FrameWidth / 10 + 23f, Game1.ScreenWidth - 50f);
    _position.Y = MathHelper.Min(_position.Y, Game1.ScreenHeight - _animations["runningR"].FrameHeight * scale);
}

/// <summary>
/// checks hit by enemy's knives.
/// </summary>

protected void CheckHealth(Sprite enemy)
{
    var halfWidth = 40 * scale;
    var halfHeight = 144 * scale;
    foreach (var enemyKnife in enemy.knivesSet)
    {
        if ((enemyKnife._position.X >= _position.X - 20) && (enemyKnife._position.X <= _position.X + 20) //sprite's body
            && (enemyKnife._position.Y >= _position.Y - 72) && enemyKnife._position.Y <= _position.Y + 72)//sprite's body
        {
            health.hit();
            enemyKnife.IsRemoved = true;
        }
    }

    if (health.rectangle.Width <= 0)// win scenario
        enemy.hasWon = true;
}
```

```
/// <summary>
///manages the movement & jumping & throwing knives of the sprite.
/// </summary>

protected virtual void Move() {

}

/// <summary>
/// sets the animations of the sprite based on Velocity, toThrow and hasJumped.
/// </summary>

protected virtual void SetAnimations()
{

}
```

- הפעולה KeepInScreen משתמשת בכלי MathHelper, שדרכו ניתן לבצע פעולות מתמטיות, ובמקרה זה שימוש בפעולה Clamp- השומרת על ערך מסוים בין טווחים מוגדרים, ושימוש בפעולה Min- שמחזירה את הערך המינימלי בין שני ערכים.
- בפעולה CheckHealth, הגדרתי בעזרת ויזואליות המשחק את הטווחים שבהם ניתן לזהות את גוף הדמות, ובכך לזהות פגיעה של סכין בדמות.
- הפעולות Move ו- SetAnimations שונות בין השחקן לבין הבוט- הרחבה בהמשך.

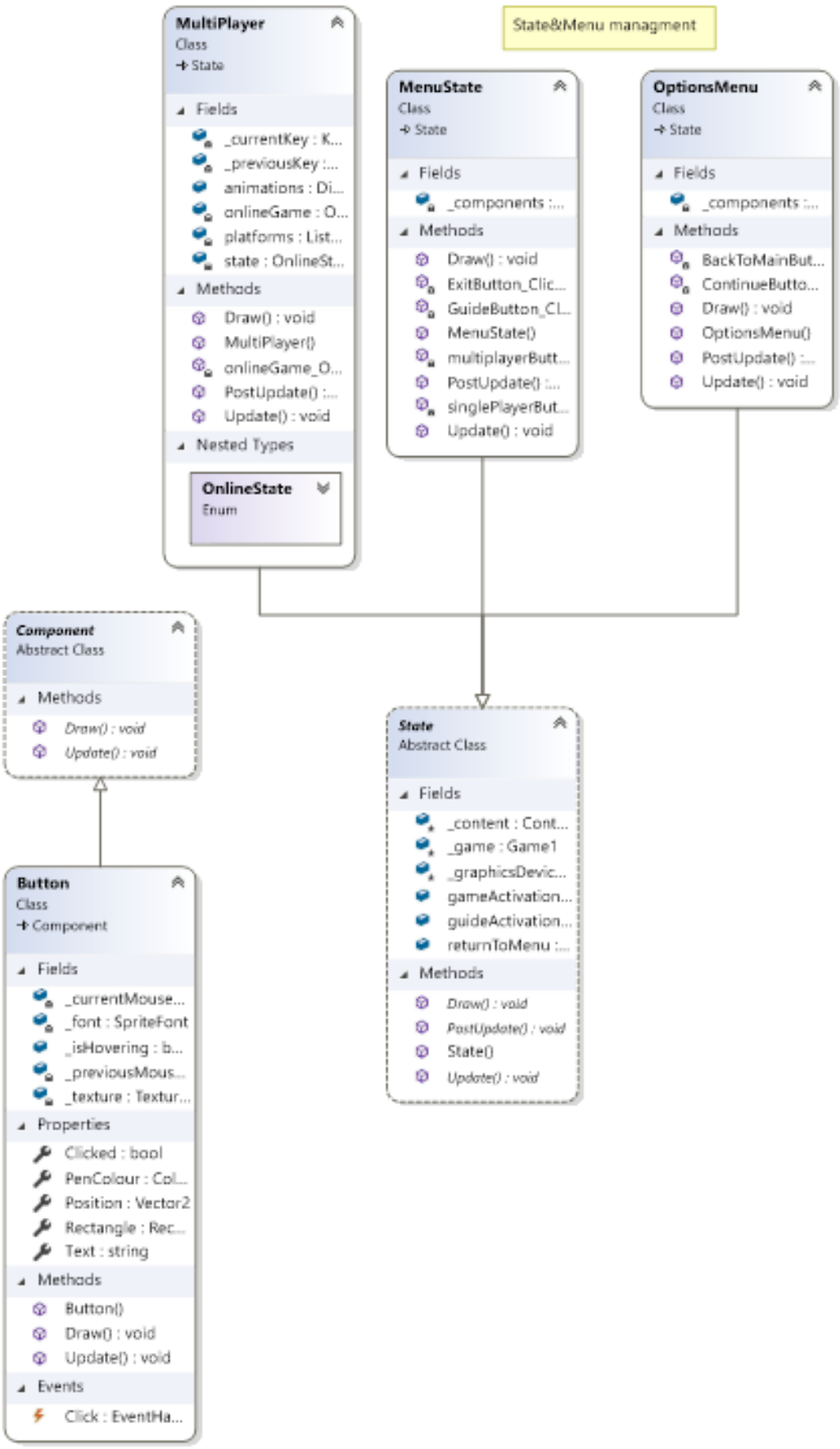
```
/// <summary>
/// adds a new knife to knivesSet
/// </summary>

protected void addKnife()
{
    Knife temp = new Knife(Game1.knifeTexture);

    temp.origin = new Vector2(Game1.knifeTexture.Width / 2, Game1.knifeTexture.Height / 2);

    temp.scale = 0.2f;

    temp.knifeDir = dir;

    temp.Position = new Vector2(_position.X, _position.Y);

    knivesSet.Add(temp);

}

/// <summary>
/// updates the knifes based on keyboard
/// </summary>

protected virtual void KnifeStates(GameTime gameTime)
{

}
```

- בפעולה AddKnife ישנה הגדרה של המשתנים שהמחלקה Knife צריכה והוספת הסכין temp לסט הסכינים באמצעות הפעולה Add.
- המחלקה KnivesStates שונה אם הדמות היא השחקן או הבוט, לכן היא ריקה ומוגדרת בתור virtual- פונקציה שהמחלקות-בת של המחלקה Sprite ישנו בהתאם לצורכיהן.

```
/// <summary>
/// updates the Sprite's animations, movement and more.
/// </summary>

public virtual void Update(GameTime gameTime, Sprite enemy)
{
    _previousKey = _currentKey;
    _currentKey = Keyboard.GetState();

    KnifeStates(gameTime);

    Move();

    SetAnimations();

    _animationManager.Update(gameTime);

    Position += Velocity;

    keepInScreen();

    CheckHealth(enemy);


}
```

- פעולה שמרכזת בתוכה קריאות לכל הפעולות לעיל, בנוסף לתחזוקת האובייקטים שמנהלים את מקשי המקלדת ועדכון מיקום הדמות בהתאם ל- Velocity.

חלק שני- ניהול כפתורים ואירועים:



המחלקה Button:

```
public class Button : Component
{
    #region Fields

    private MouseState _currentMouse;// mouse managment

    private MouseState _previousMouse;// mouse managment

    private SpriteFont _font;

    public bool _isHovering;//if the mouse is on the button

    private Texture2D _texture;

    #endregion

    #region Properties

    public event EventHandler Click;// Event that is called once a click occours.

    public bool Clicked { get; private set; }// if the button is clicked or not

    public Color PenColour { get; set; }

    public Vector2 Position { get; set; }

    public Rectangle Rectangle
    {
        get
        {
            return new Rectangle((int)Position.X, (int)Position.Y, _texture.Width, _texture.Height);
        }
    }
```

```
public string Text { get; set; }

#endregion

#region Methods

/// <summary>
/// Button's constructor
/// </summary>

public Button(Texture2D texture, SpriteFont font)
{
    _texture = texture;

    _font = font;

    PenColour = Color.Black;
}
```

- בפעולה Button נמצאות ההגדרות של כפתור במשחק- ציור שלו, עדכון שלו והגדרה של מה קורה כאשר לוחצים על הכפתור.
- באזור ה- data של המחלקה יש שימוש באובייקט EventHandler, אשר מנהל יצירת אירועים, קריאה שלהם ב-namespace וקריאה לפעולות כתוצאה מכך.

```
/// <summary>
/// draws the button with text in it.
/// </summary>

public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
    var colour = Color.White;

    if (_isHovering)
    {
        colour = Color.Gray;
    }

    spriteBatch.Draw(_texture, Rectangle, colour);

    if (!string.IsNullOrEmpty(Text))// if Text contains text, draw it inside the button.
    {
        var x = (Rectangle.X + (Rectangle.Width / 2)) - (_font.MeasureString(Text).X / 2);
        var y = (Rectangle.Y + (Rectangle.Height / 2)) - (_font.MeasureString(Text).Y / 2);

        spriteBatch.DrawString(_font, Text, new Vector2(x, y), PenColour);
    }
}
```

- בפעולה Draw ישנה התייחסות לעליית העכבר על הכפתור, ובמקרה זה צבע הכפתור יתחלף לאפור. בנוסף, ישנו תנאי שמונע קריסה של הפעולה במקרה ש- Text ריק או שווה ל- "". הטקסט מודפס במרכז הכפתור.

```
/// <summary>
/// checks mouse hovering and call the event in case of clicking.
/// </summary>

public override void Update(GameTime gameTime)
{
    _previousMouse = _currentMouse;
    _currentMouse = Mouse.GetState();

    var mouseRectangle = new Rectangle(_currentMouse.X, _currentMouse.Y, 1, 1);
    var prevMouseRec = new Rectangle(_previousMouse.X, _previousMouse.Y, 1, 1);

    _isHovering = false;

    if (mouseRectangle.Intersects(Rectangle))// hover
    {
        _isHovering = true;

        if (_currentMouse.LeftButton == ButtonState.Released && _previousMouse.LeftButton == ButtonState.Pressed)// click
        {
            Game1.click.Play();
            Click?.Invoke(this, new EventArgs());
        }
    }

    if (mouseRectangle.Intersects(Rectangle) && !prevMouseRec.Intersects(Rectangle))
    {
        Game1.hovering.Play();
    }
}
```

- בעזרת עטיפת הקצה של העכבר במלבן בגודל פיקסל אחד, ניתן לזהות בקלות עלייה של העכבר על הכפתור- באמצעות Rectangle.Intersects. השורה:

```
Click?.Invoke(this, new EventArgs());
```

אומרת- אם ה- EventHandler מסוג קליק אינו שווה null, בצע קריאה מסוג Click ברחבי ה- namespace.

- בנוסף יש העברה של קבצי audio מהמחלקה הראשית לפעולה זו, מכיוון שהם נטענים שם.

המחלקה State:

```
public abstract class State
{

    public bool gameActivation = false;// for Game1
    public bool returnToMenu = false;// for Game1
    public bool guideActivation = false;// for Game1

    #region fields

    protected ContentManager _content;

    protected GraphicsDevice _graphicsDevice;

    protected Game1 _game;

    #endregion

    #region Methods

    /// <summary>
    /// regular draw function
    /// </summary>

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

    /// <summary>
    /// State's Constructor
    /// </summary>

    public State(Game1 game, GraphicsDevice graphicsDevice, ContentManager content)
    {
        _game = game;

        _graphicsDevice = graphicsDevice;

        _content = content;
    }

    /// <summary>
    /// regular update function
    /// </summary>

    public abstract void Update(GameTime gameTime);


    #endregment
}
```

- המחלקה היא אבסטרקטית, כלומר היא מתווה את הכותרת של הפעולות האבסטרקטיות- שם הפעולה, סוגה והמשתנים שהיא מקבלת.
- התכונות של הפעולה נועדו לדמות פעילות של Game1, כלומר ניהול משחק בעזרת גישה לתיקיית content, שימוש בכלים גרפיים ובפעולות של מחלקת Game.

המחלקה MenuState- תפריט הפתיחה:

```
public class MenuState : State
{
    private List<Component> _components;// contains the buttons

    /// <summary>
    /// MenuState's constructor.
    /// </summary>
    public MenuState(Game1 game, GraphicsDevice graphicsDevice, ContentManager content) : base(game, graphicsDevice, content)
    {
        #region Creating the buttons
        var buttonTexture = _content.Load<Texture2D>("Controls/blueButton");

        var positionX = (Game1.ScreenWidth / 2) - (buttonTexture.Width / 2);

        var buttonFont = _content.Load<SpriteFont>("Fonts/Font");

        var singleplayerButton = new Button(buttonTexture, buttonFont)
        {
            Position = new Vector2(positionX, 200),
            Text = "Singleplayer",
        };

        singleplayerButton.Click += singlePlayerButton_Click;// adding a function to the event of the particular button.

        var multiplayerButton = new Button(buttonTexture, buttonFont)
        {
            Position = new Vector2(positionX, 300),
            Text = "Multiplayer",
        };

        multiplayerButton.Click += multiplayerButton_Click;// adding a function to the event of the particular button.

        var guideButton = new Button(buttonTexture, buttonFont)
        {
            Position = new Vector2(positionX, 400),
            Text = "How to Play?",
        };
        guideButton.Click += GuideButton_Click;// adding a function to the event of the particular button.

        var exitButton = new Button(buttonTexture, buttonFont)
        {
            Position = new Vector2(positionX, 500),
            Text = "Exit"
        };
        exitButton.Click += ExitButton_Click;// adding a function to the event of the particular button.

        _components = new List<Component>()// filling the button list
        {
            singleplayerButton,
            multiplayerButton,
            guideButton,
            exitButton,

        };
        #endregion
    }
```

- המחלקה של תפריט הפתיחה של המשחק.
- יצירת הכפתורים והגדרת פעולות שיקרו ברגע שה- Event יוכרז.

```
private void ExitButton_Click(object sender, EventArgs e)
{
    _game.Exit();// closing the game.
}

private void GuideButton_Click(object sender, EventArgs e)
{
    guideActivation = true;// opening the guide in Game1.
}

/// <summary>
/// drawing the buttons.
/// </summary>
public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{

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

}

/// <summary>
/// changing the state to multiplayer.
/// </summary>

private void multiplayerButton_Click(object sender, EventArgs e)
{
    _game.ChangeState(new MultiPlayer(_game, _graphicsDevice, _content));
}

/// <summary>
/// audio and boolean managment in Game1.
/// </summary>

private void singlePlayerButton_Click(object sender, EventArgs e)
{
    MediaPlayer.Stop();
    MediaPlayer.Play(Game1._duelSong);
    gameActivation = true;
    Game1.activeGame = true;
    Game1.escPress = false;
}

/// <summary>
/// updating the buttons.
/// </summary>

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

}
```

- הפעולות שקורות ברגע הלחיצה , ציור ועדכון הכפתורים.

המחלקה OptionsMenu- תפריט העצירה:

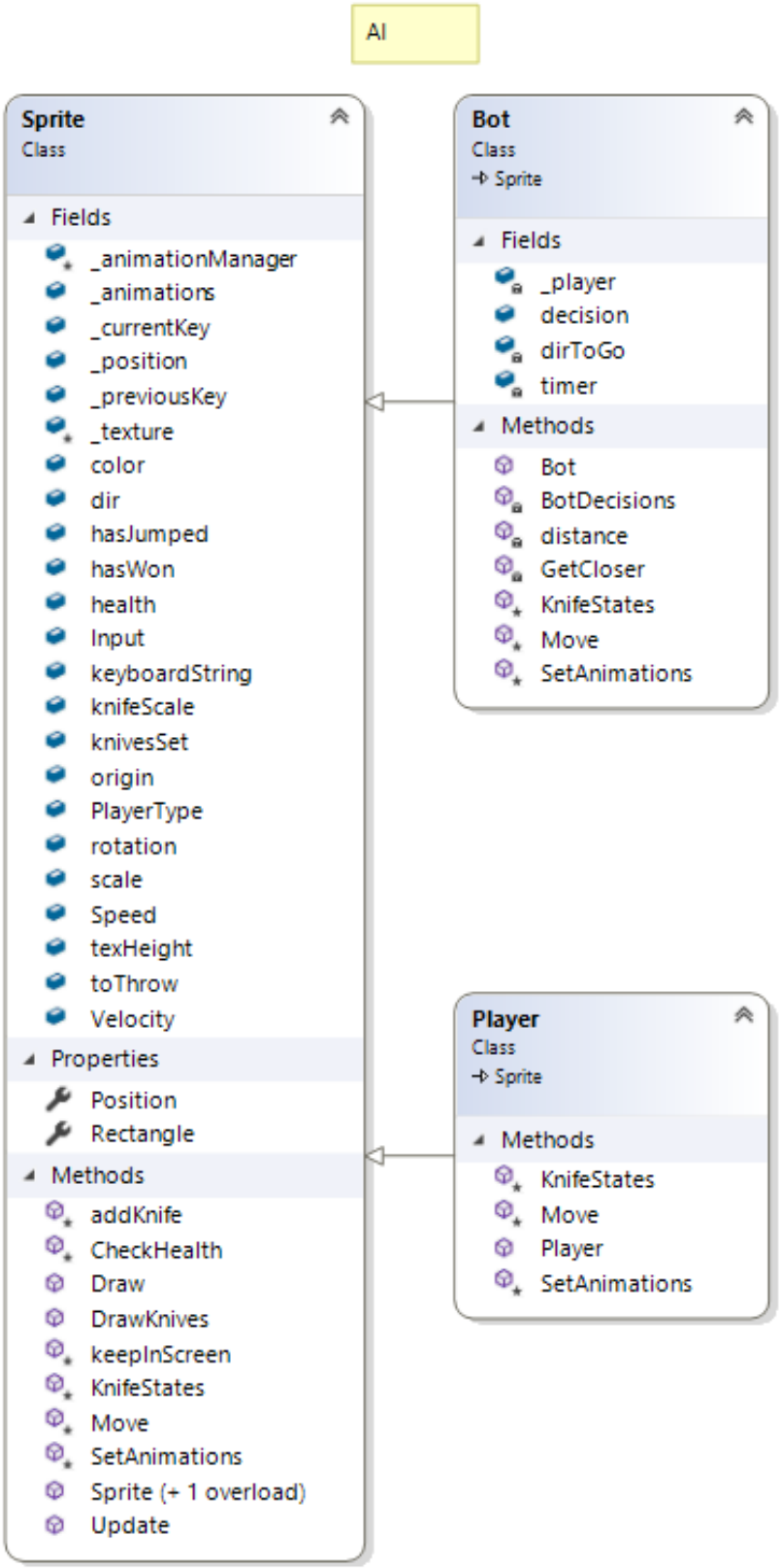
```
public class OptionsMenu : State
{
    private List<Component> _components;// contains the buttons

    /// <summary>
    /// OptionsMenu's constructor.
    /// </summary>

    public OptionsMenu(Game1 game, GraphicsDevice graphicsDevice, ContentManager content) : base(game, graphicsDevice, content)
    {
        var buttonTexture = _content.Load<Texture2D>("Controls/wooden button");

        var positionX = (Game1.ScreenWidth / 2) - (buttonTexture.Width / 2);

        var buttonFont = _content.Load<SpriteFont>("Fonts/Font");

        var ContinueButton = new Button(buttonTexture, buttonFont)
        {
            Position = new Vector2(positionX, 200),
            Text = "Continue",
        };
        ContinueButton.Click += ContinueButton_Click;// return back to game

        var BackToMainButton = new Button(buttonTexture, buttonFont)
        {
            Position = new Vector2(positionX, 300),
            Text = "Main Menu",
        };
        BackToMainButton.Click += BackToMainButton_Click;// returns to main menu.

        _components = new List<Component>()//filling up the list.
        {
            ContinueButton,
            BackToMainButton,
        };

    }
```

- המחלקה של התפריט שנפתח כאשר נעצר המשחק.
- יצירת הכפתורים והגדרת הפעולות שמתרחשות בעת הקריאה של ה- Event.

```
/// <summary>
/// managing Game1 booleans and playing again the opening music.
/// </summary>
private void BackToMainButton_Click(object sender, EventArgs e)
{
    returnToMenu = true;
    Game1.activeGame = true;
    MediaPlayer.Play(Game1.menuSong);
}

/// <summary>
/// managing Game1 booleans and playing again the duel music.
/// </summary>

private void ContinueButton_Click(object sender, EventArgs e)
{
    Game1.escPress = false;
    Game1.activeGame = true;
    MediaPlayer.Play(Game1._duelSong);
}

/// <summary>
/// drawing the buttons.
/// </summary>

public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
    foreach (var component in _components)
        component.Draw(gameTime, spriteBatch);
}

/// <summary>
/// updating the buttons.
/// </summary>

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

- הגדרת תוכן הפעולות של האירועים, עדכון וציור הכפתורים.

חלק שלישי- הבינה המלאכותית:



המחלקה Player- השחקן:

```
class Player : Sprite
{
    /// <summary>
    /// Player's construcrtor.
    /// </summary>
    public Player(Dictionary<string, Animation> animations) : base(animations)
    {
        PlayerType = "player";
        health = new HealthBar(Game1.healthTex, new Vector2(119, 101),
              new Rectangle(0, 0, Game1.healthTex.Width - 20, Game1.healthTex.Height - 4));

        knivesSet = new List<Knife>();
        Position = new Vector2(50, 700);
        scale = 0.35f;
        knifeScale = 0.15f;
        Speed = 5f;
        origin = new Vector2(Game1.anyPlayerTex.Width / 20, Game1.anyPlayerTex.Height / 2);
        texHeight = Game1.anyPlayerTex.Height * 0.35f;
        Input = new Input()
        {
            Down = Keys.S,
            Left = Keys.A,
            Right = Keys.D,
            Throw = Keys.E,
            Jump = Keys.Space,
        };
        color = Color.White;
    }
```

- המחלקה Player יורשת מהמחלקה Sprite, ובכך יכולה להשתמש בפעולות ובתכונות שלה. בפעולה הבונה של Player, מופיעה הפקודה `: base(animations)` שאומרת כך- שורות הקוד שבתוך הפעולה הבונה של Player (המחלקה-בת) יבוצעו, ובסופן שורות הקוד של הפעולה הבונה של Sprite (המחלקה-אם) יבוצעו.
- בתוך הפעולה הבונה של Player, משתנים בסיסיים של הדמות מוגדרים- scale, speed, color ועוד. בנוסף, יש משתנים שייחודיים רק לשחקן- המיקום ההתחלתי שלו על המסך והגדרת ה- input- באמצעותו השחקן זז באמצעות מקשי המקלדת.

```
/// <summary>
/// oerrides the Move func in Sprite class
/// </summary>
protected override void Move()
{
    if (Keyboard.GetState().IsKeyDown(Input.Right))
    {
        Velocity.X = Speed;// moves right.
        dir = "right";
        keyboardString = "right";
    }
    else if (Keyboard.GetState().IsKeyDown(Input.Left))
    {
        Velocity.X = -Speed;// moves left.
        dir = "left";
        keyboardString = "left";
    }
    else Velocity.X = 0f;//stands still.

    #region Jumping

    if (Keyboard.GetState().IsKeyDown(Input.Jump) && hasJumped == false)//jumping scenario.
    {
        _position.Y -= 20f;
        Velocity.Y = -10f;
        hasJumped = true;
        keyboardString = "jump";
    }

    float i = 1;
    Velocity.Y += 0.15f * i;// makes the sprite fall down.


    if (_position.Y + texHeight >= Game1.ScreenHeight)// stop falling when hits ground.
    {
        hasJumped = false;
    }

    if (hasJumped == false)// stop falling when hits ground.
    {
        Velocity.Y = 0f;
    }

    #endregion
    if (Keyboard.GetState().IsKeyDown(Input.Throw))// throwing scenario
    {
        toThrow = true;
        keyboardString = "throw";
    }

    if (Keyboard.GetState().IsKeyDown(Input.Down))
    {
        keyboardString = "down";
    }
}
```

- המשתנה הבוליאני toThrow מנהל את זריקת הסכין- אם נלחץ על המקש של הזריקה, הוא יהיה true והפעולות של יצירת ועדכון הסכין יחלו.
- השדה Velocity מסוג Vector2 מכיל את הכיוון שהמהירות של הדמות- בהתאם ללחיצה על המקשים ימינה ושמאלה, ערך ה-X ישתנה כפי שמתואר בקוד.
- במצב של קפיצה, ערך ה-Y של ה- Velocity נהיה שלילי בזמן שערך ה-Y של מיקום הדמות יורד בפתאומיות, שיוצרת את הקפיצה. לאחר מכן, ערך ה-Y של מיקום הדמות עולה בהדרגה עד שהדמות מגיעה לרצפה.

```
/// <summary>
/// overrides the SetAnimations func in Sprite class.
/// </summary>
protected override void SetAnimations()
{

    if (Velocity.X == 0f && Velocity.Y == 0f)
    { //displaying the direction of the standing sprite.
        if (dir.Equals("none"))
            _animationManager.Play(_animations["standingR"]);
        else if (dir.Equals("right"))
            _animationManager.Play(_animations["standingR"]);
        else if (dir.Equals("left"))
            _animationManager.Play(_animations["standingL"]);

        if (toThrow)// throwing animation
        {
            if (dir.Equals("right"))
                _animationManager.Play(_animations["throwingR"]);
            else if (dir.Equals("left"))
                _animationManager.Play(_animations["throwingL"]);
        }
    }

    if (toThrow)// throwing animation
    {
        if (dir.Equals("right"))
            _animationManager.Play(_animations["throwingR"]);
        else if (dir.Equals("left"))
            _animationManager.Play(_animations["throwingL"]);
    }

    if (!Keyboard.GetState().IsKeyUp(Input.Throw))// preventing spam clicking.
    {
        toThrow = false;
        keyboardString = "";
    }

    if (!hasJumped && Velocity.X > 0 && keyboardString.Equals("right"))
        _animationManager.Play(_animations["runningR"]);

    else if (!hasJumped && Velocity.X < 0 && keyboardString.Equals("left"))
        _animationManager.Play(_animations["runningL"]);


    else if (hasJumped)// jumping animation
    {
        if (Velocity.X > 0)
            _animationManager.Play(_animations["jumpingR"]);
        else if (Velocity.X < 0) _animationManager.Play(_animations["jumpingL"]);
    }
}
```

- בהתאם לכיוון ול- Velocity של הדמות, אובייקט ניהול האנימציה (AnimationManager) קורא לפעולה Play ומספק לה סט אחר של אנימציות. כדי להגיע לערך מסוים במילון, יש לכתוב באופן הבא:

```
_animationManager.Play(_animations["standingR"]);
```

- במצב של זריקת סכין, המשתנה direction חשוב במיוחד- הוא קובע לאיזה כיוון הסכין תזוז, ואיזה אנימציה תפעל על הדמות.

התנאי: `if (!Keyboard.GetState().IsKeyUp(Keys.E))` מונע מצב שבו השחקן ילחץ לחיצה ארוכה על המקש ויזרוק רצף סכינים- על השחקן ללחוץ ולשחרר את המקש כדי שהזריקה תקרה.

```
/// <summary>
/// overrides the KnifeStates func in Sprite Class.
/// </summary>
protected override void KnifeStates(GameTime gameTime)
{
    if (_previousKey.IsKeyDown(Input.Throw) &&
            _currentKey.IsKeyUp(Input.Throw))
    {
        addKnife();
        Game1.throwing.Play(volume: 0.2f, pitch: 0.0f, pan: 0.0f);
    }

    foreach (var knife in knivesSet)
    {
        knife.UpdateKnife(gameTime);
    }
}
```

- כאשר מקש הזריקה נלחץ (פעם אחת), סכין מתווספת לסט הסכינים וסאונד של זריקת סכין מושמע.
- לאחר מכן ישנה לולאה שעוברת על הסכינים בסט ומעדכנת אותם- פירוט במחלקה Knife.

המחלקה Bot- האויב של השחקן:

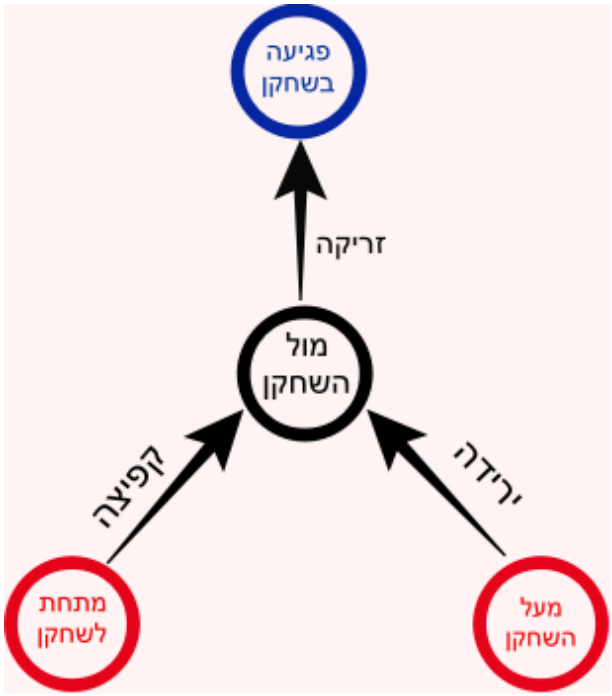
```
class Bot : Sprite
{
    Player _player;

    public string decision = ""; //what the bot is going to do.
    string dirToGo = "";// what direction the bot needs to would go to.

    float timer;

    /// <summary>
    /// Bot's constructor.
    /// </summary>
    public Bot(Dictionary<string, Animation> animations, Player player) : base(animations)
    {
        _player = player;
        PlayerType = "enemy";
        health = new HealthBar(Game1.healthTex, new Vector2(1019, 101),
                new Rectangle(0, 0, Game1.healthTex.Width - 20, Game1.healthTex.Height - 4));
        Position = new Vector2(1200, 700);
        knivesSet = new List<Knife>();
        scale = 0.35f;
        Speed = 3f;
        knifeScale = 0.15f;
        origin = new Vector2(Game1.anyPlayerTex.Width / 20, Game1.anyPlayerTex.Height / 2);
        texHeight = Game1.anyPlayerTex.Height * 0.35f;
        color = new Color(64, 13, 13);
    }
```

- המחלקה של הבוט שנלחם נגד השחקן במשחק.
- בדומה ל-Player, הפעולה הבונה של Bot מכילה שורות קוד, בנוסף לפעולה הבונה של Sprite. ההבדלים בין הבוט לשחקן הם המיקום ההתחלתי של כל אחד מהם (מתחילים בקצוות השונים של המסך), והגורם שמשפיע על התזוזה שלהם (אצל השחקן זה ה- input ואצל הבוט זה המשתנה decision).
- הפעולה הבונה של Bot מקבלת את השחקן השני בכותרת הפעולה, כדי לבצע פעולות בהתאם למיקום השחקן.

```
/// <summary>
/// returns the distance between the two players.
/// </summary>
private float distance()
{
    return Math.Abs(_position.X - _player._position.X);
}

/// <summary>
/// decides what direction the bot in facing.
/// </summary>
private void GetCloser()
{
    if (_position.X > _player._position.X)
    {
        dirToGo = "left";
    }
    else
    {
        dirToGo = "right";
    }

}
```

- שתי פעולות עזר:
- Distance: מחשבת את המרחק בין השחקן לבוט, תוך שימוש בפעולה Abs של המחלקה Math- שמקבלת ערך מסוים ומחזירה את הערך המוחלט שלו.
- GetCloser: מחליטה באמצעות המשתנה dirToGo לאיזה כיוון הבוט יסתכל.

```
/// <summary>
/// determines what the bot should do based on the player's position.
/// </summary>
private void BotDecisions()
{
    if((_position.Y >= _player._position.Y - 72) && (_position.Y <= _player._position.Y + 72)){//in front of player
        decision = "throw";

        if (_position.X > _player._position.X)// to face the player
        {
            dir = "left";
        }
        else
        {
            dir = "right";
        }
    }

    else if(_position.Y > _player._position.Y) // beneath the player
    {
        decision = "jump";
    }
    else // above the players.
    {
        decision = "down";
    }

    if(distance() >= 350)// the gap between bot & player.
    {
        GetCloser();
    }
    else// stops the bot's movement.
    {
        dirToGo = "";
    }

}
```



- פעולה שמחליטה כיצד הבוט יפעל:
- בהתחלה, היא בודקת האם הבוט נמצא באותו גובה כמו השחקן- על הרצפה, על פלטפורמה או באוויר. אם כן, שיעמוד מול השחקן ויתחיל לזרוק סכינים.
- אם לא, שיקפוץ או ירד למטה כדי להגיע למצב שבו הוא עומד מול השחקן.
- הגדרתי מרחק קבוע בין השחקן לבוט שבו הבוט יעצור ויתחיל לזרוק. זוהי החלטה אישית בהתאם לנראות על המסך.
- אם הבוט רחוק יותר מהשחקן, ישנה קריאה לפעולה GetCloser עליה פירטתי לעיל.

```
/// <summary>
/// overrides the Move func in Sprite class.
/// </summary>
protected override void Move()
{
    BotDecisions();
    if (dirToGo.Equals("right"))
    {
        Velocity.X = Speed;// moves right.
        dir = "right";
        keyboardString = "right";
    }
    else if (dirToGo.Equals("left"))
    {
        Velocity.X = -Speed;// moves left.
        dir = "left";
        keyboardString = "left";
    }
    else Velocity.X = 0f;//stands still.

    #region Jumping

    if (decision.Equals("jump") && hasJumped == false)//jumping scenario.
    {
        _position.Y -= 20f;
        Velocity.Y = -10f;
        hasJumped = true;
        keyboardString = "jump";
    }

    float i = 1;
    Velocity.Y += 0.15f * i;// makes the sprite fall down.


    if (_position.Y + texHeight >= Game1.ScreenHeight)// stop falling when hits ground.
    {
        hasJumped = false;
    }

    if (hasJumped == false)// stop falling when hits ground.
    {
        Velocity.Y = 0f;
    }

    #endregion

    if (decision.Equals("throw"))// throwing scenario
    {
        toThrow = true;
        keyboardString = "throw";
    }

    if (decision.Equals("down"))
    {
        keyboardString = "down";
    }
}
```

- פעולת Move דומה לזאת של Player, אך כאן ניהול התנועה של הבוט מתבצע בהתאם למשתנה decision, שמוגדר בפעולה BotDecisions אליה יש קריאה בתחילת הפעולה.

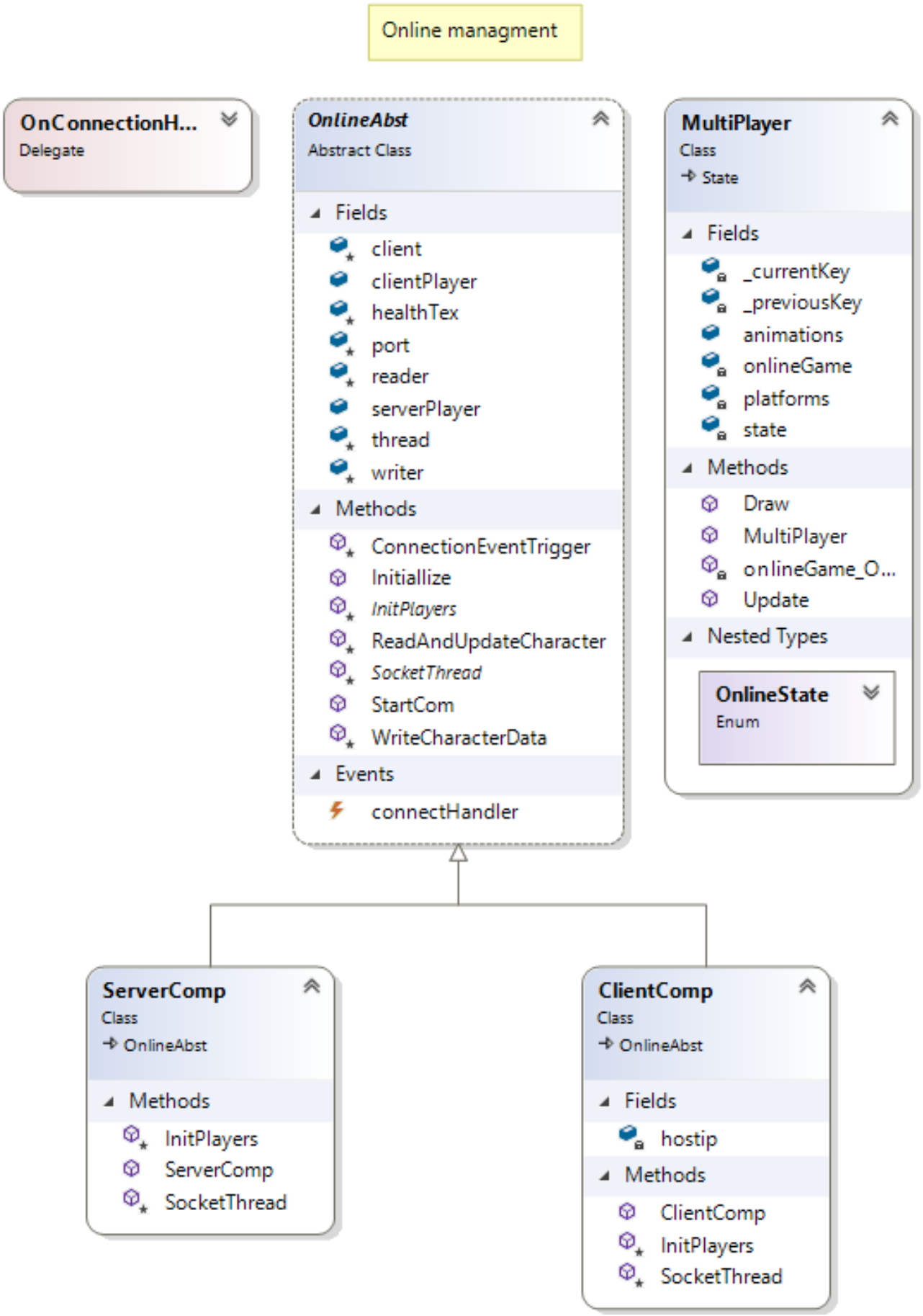
```
/// <summary>
/// overrides the SetAnimations in Sprite class.
/// </summary>
protected override void SetAnimations()
{
    if (Velocity.X == 0f && Velocity.Y == 0f)
    { //displaying the direction of the standing sprite.
        if (dir.Equals("none"))
            _animationManager.Play(_animations["standingR"]);
        else if (dir.Equals("right"))
            _animationManager.Play(_animations["standingR"]);
        else if (dir.Equals("left"))
            _animationManager.Play(_animations["standingL"]);

        if (toThrow)// throwing animation
        {
            if (dir.Equals("right"))
                _animationManager.Play(_animations["throwingR"]);
            else if (dir.Equals("left"))
                _animationManager.Play(_animations["throwingL"]);
        }
    }

    if (toThrow)// throwing animation
    {
        if (dir.Equals("right"))
            _animationManager.Play(_animations["throwingR"]);
        else if (dir.Equals("left"))
            _animationManager.Play(_animations["throwingL"]);
    }

    if (dirToGo.Equals("right"))
        _animationManager.Play(_animations["runningR"]);

    else if (dirToGo.Equals("left"))
        _animationManager.Play(_animations["runningL"]);


    else if (hasJumped)// jumping animation
    {
        if (Velocity.X > 0)
            _animationManager.Play(_animations["jumpingR"]);
        else if (Velocity.X < 0) _animationManager.Play(_animations["jumpingL"]);
    }
}
```

- מגדיר את האנימציות של הבוט, באמצעות המשתנים שיש ב- Sprite ובאמצעות המשתנה decision.

```
/// <summary>
/// overrides the KnifeStates in Sprite class.
/// </summary>
protected override void KnifeStates(GameTime gameTime)
{
    timer += (float)gameTime.ElapsedGameTime.TotalSeconds;

    if(timer >= 0.85)// sets difficulty of game.
    {
        if (decision.Equals("throw"))
        {
            addKnife();
            toThrow = false;
            Game1.throwing.Play(volume: 0.2f, pitch: 0.0f, pan: 0.0f);
        }
        decision = "";
        timer = 0;
    }
    foreach (var knife in knivesSet)
    {
        knife.UpdateKnife(gameTime);
    }
}
```

- שימוש במשתנה של טיימר שקובע באיזה קצב הבוט יזרוק את הסכין על השחקן, כאשר ניתן להגדיר את הקצב ובכך את קושי המשחק.
- בהנחה שזמן זה עבר, מתווספת סכין לסט ויש סאונד של זריקת סכין.
- כל הסכינים שבסט של הבוט מתעדכנים בכל מקרה.

חלק רביעי- אינטרנט ותקשורת

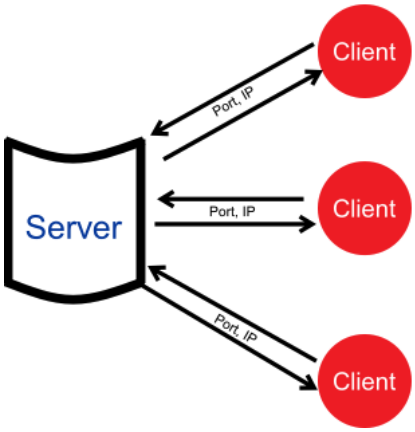


המחלקה OnlineAbst:

```
public delegate void OnConnectionHandler(); // creates event.

abstract class OnlineAbst
{
    #region properties
    public event OnConnectionHandler connectHandler; // raises on connection.

    protected BinaryReader reader;// translates the binary code into useable information.
    protected BinaryWriter writer; // transforms the information to binary code.

    protected int port;

    protected Thread thread; // secondary thread to manage the communications.

    protected TcpClient client; // gets the bit stream.

    public Sprite serverPlayer, clientPlayer; // the two players in the game.

    protected Texture2D healthTex; // for the sprite constructor.


    #endregion

    #region Methods

    /// <summary>
    /// raises when the func is called
    /// </summary>
    protected void ConnectionEventTrigger()
    {
        connectHandler?.Invoke();
    }
```

- המחלקה OnlineAbst היא מחלקה אבסטרקטית שהמחלקות הבאות יורשות ממנה.
- במחלקה זו מוגדרות הפעולות שקורות כדי ליצור חיבור בין השרת ללקוח, ומה קורה כאשר ישנו חיבור בין השרת ללקוח.
- כדי ליצור את החיבור, ישנו שימוש ב- Thread (תהליכון)- ניהול משני של המשחק שבתוכו מנוהל החיבור והתקשורת בין השרת ללקוחות שונים. התהליכון הראשי והמשני פועלים במקביל, כך שריצת המשחק חלקה ומהירות התגובה מהירה.
- המשתנה client הינו מסוג TCPClient, מחלקה המכילה את פרוטוקול TCP עליו מפורט בפרק "מושגים המוזכרים בעבודה".
- הפעולה ConnectionEventTrigger מפעילה את ה- event.

```
/// <summary>
/// called when the connection starts.
/// </summary>
public void Initiallize(Dictionary<string, Animation> animations, ContentManager content)
{
    InitPlayers(animations, content);
    StartCom();
}

/// <summary>
/// gets the bit stream and sets the character by it.
/// </summary>
protected void ReadAndUpdateCharacter(Sprite character)
{
    character.Velocity.X = reader.ReadSingle();
    character.Velocity.Y = reader.ReadSingle();
    character._position.X = reader.ReadSingle();
    character._position.Y = reader.ReadSingle();
    character.hasJumped = reader.ReadBoolean();
    character.toThrow = reader.ReadBoolean();
    character.dir = reader.ReadString();
}

/// <summary>
/// sends a bit stream of the variables needed in connection.
/// </summary>
protected void WriteCharacterData(Sprite character)
{
    writer.Write(character.Velocity.X);
    writer.Write(character.Velocity.Y);
    writer.Write(character._position.X);
    writer.Write(character._position.X);
    writer.Write(character.hasJumped);
    writer.Write(character.toThrow);
    writer.Write(character.dir);
}
```



- הפעולה Initialize מאפסת את הדמויות ויוצרת את החיבור והתקשורת בין שני הגופים.
- הפעולות הבאות עוסקות בתקשורת בין שני הגופים:
- ReadAndUpdateCharacter מקבלת זרם של רצפים של ביטים בתקשורת, ממירה אותם לסוגים שונים של מידע- משתנה בוליאני, מחרוזת או מספר מכל סוג, ומכניסה אותם לתכונות של character, כך שהוא יתעדכן בהתאם למידע שנקלט מהתקשורת.
- WriteCharacterData מעבירה זרם של ביטים בתקשורת באמצעות הפעולה Write של המחלקה BinaryWriter. הזרם מכיל רצפים שונים של ביטים, כאשר כל אחד מהם מייצג משתנה של המחלקה Sprite.

```
/// <summary>
/// connection managments in derived classes.
/// </summary>
protected abstract void SocketThread();

/// <summary>
/// creates the players in derived classes.
/// </summary>
protected abstract void InitPlayers(Dictionary<string, Animation> animations, ContentManager content);

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

- SocketThread- פעולה אבסטרקטית שפותחת את הקשר בין השרת ללקוח באמצעות ה- Thread ומנהלת את העברת הנתונים.
- InitPlayers- פעולה אבסטרקטית שבה הדמויות נוצרות- בפעולה אצל השרת דמות אחת היא השחקן והשנייה היא האויב, לעומת הפעולה אצל הלקוח בה זה הפוך.
- StartCom- יוצר ופותח את ה- Thread ומגדיר אותו כזמין ליצירת תקשורת בשורה `thread.Start();`.

המחלקה ServerComp:

```
class ServerComp : OnlineAbst
{
    /// <summary>
    /// ServerComp's constructor.
    /// </summary>
    public ServerComp(int port)
    {
        this.port = port;
    }

    /// <summary>
    /// creates the characters- player and enemy.
    /// </summary>
    protected override void InitPlayers(Dictionary<string, Animation> animations, ContentManager content)
    {
        healthTex = content.Load<Texture2D>("Objects/HealthBarInside");

        serverPlayer = new Sprite(animations)
        {
            PlayerType = "player",
            health = new HealthBar(healthTex, new Vector2(119, 101),
                  new Rectangle(0, 0, healthTex.Width - 20, healthTex.Height - 4)),

            knivesSet = new List<Knife>(),
            Position = new Vector2(50, 700),
            scale = 0.35f,
            knifeScale = 0.15f,
            origin = new Vector2(Game1.anyPlayerTex.Width / 20, Game1.anyPlayerTex.Height / 2),
            texHeight = Game1.anyPlayerTex.Height * 0.35f,
            color = Color.White,
        };

        clientPlayer = new Sprite(animations)
        {
            PlayerType = "enemy",
            health = new HealthBar(healthTex, new Vector2(1019, 101),
                new Rectangle(0, 0, healthTex.Width - 20, healthTex.Height - 4)),
            Position = new Vector2(1200, 700),
            knivesSet = new List<Knife>(),
            scale = 0.35f,
            knifeScale = 0.15f,
            origin = new Vector2(Game1.anyPlayerTex.Width / 20, Game1.anyPlayerTex.Height / 2),
            texHeight = Game1.anyPlayerTex.Height * 0.35f,
            Input = new Input(),
            color = new Color(64, 13, 13),
        };
    }
```

- המחלקה של השרת במשחק- קבלת מידע מהלקוחות ויצירת התהליכון.
- המחלקה יורשת מהמחלקה OnlineAbst, והפעולה InitPlayers דורסת את זאת של מחלקת האם. פעולה זו מאתחלת את הדמויות ומגדירה את הערכים הראשוניים שלהם.
- ה- serverPlayer מוגדר כשחקן הנוכחי, וה- clientPlayer מוגדר כאויב.
- port= כלי שבעזרתו מבצעים העברת נתונים באופן ישיר בין כלי תקשורת.

```
    /// <summary>
    /// creates the connection and receives data from clients in a secondary thread.
    /// </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());

        ConnectionEventTrigger();

        while (true)
        {
            WriteCharacterData(serverPlayer);
            ReadAndUpdateCharacter(clientPlayer);
        }
    }
}
```

- הפעולה SocketThread דורסת את זאת של המחלקה OnlineAbst, ממנה היא יורשת.
- מטרת הפעולה היא יצירת מצב של המתנה על קבלת מידע מן הלקוחות דרך זרם של רצפי ביטים המעבירים נתונים. הפקודה `client.GetStream()` נותנת גישה לרצף הביטים שבאמצעותו עוברת התקשורת, כאשר האובייקטים reader ו- writer משתמשים בזרם זה למטרות שונות, כפי שמפורט בתיעוד המחלקה OnlineAbst.
- בפעולה קיימת גם קריאה ל- event שהוגדר במחלקת האם, שברגע החיבור יהיה ניתן לזהותו ברחבי ה- namespace.
- בסוף הפעולה, ישנה לולאה אינסופית שמטרתה פשוטה- כאשר ייווצר החיבור, זרם הביטים של השרת ללקוח יעבוד תמיד, עד שבחיבור נקטע ברגע סגירת המשחק.

המחלקה ClientComp:

```
class ClientComp : OnlineAbst
{
    string hostip; // to find the server.

    /// <summary>
    /// ClientComp's constructor.
    /// </summary>
    public ClientComp(string hostip, int port)
    {
        this.port = port;
        this.hostip = hostip;

    }

    /// <summary>
    /// creates the characters- player and enemy.
    /// </summary>
    protected override void InitPlayers(Dictionary<string, Animation> animations, ContentManager content)
    {
        healthTex = content.Load<Texture2D>("Objects/HealthBarInside");

        serverPlayer = new Sprite(animations)
        {
            PlayerType = "enemy",
            health = new HealthBar(healthTex, new Vector2(1019, 101),
                new Rectangle(0, 0, healthTex.Width - 20, healthTex.Height - 4)),
            _position = new Vector2(50, 700),
            knivesSet = new List<Knife>(),
            scale = 0.35f,
            knifeScale = 0.15f,
            origin = new Vector2(Game1.anyPlayerTex.Width / 20, Game1.anyPlayerTex.Height / 2),
            texHeight = Game1.anyPlayerTex.Height * 0.35f,
            color = new Color(64, 13, 13),

        };

        clientPlayer = new Sprite(animations)
        {
            PlayerType = "player",
            health = new HealthBar(healthTex, new Vector2(119, 101),
                  new Rectangle(0, 0, healthTex.Width - 20, healthTex.Height - 4)),

            knivesSet = new List<Knife>(),
            _position = new Vector2(1200, 700),
            scale = 0.35f,
            knifeScale = 0.15f,
            origin = new Vector2(Game1.anyPlayerTex.Width / 20, Game1.anyPlayerTex.Height / 2),
            texHeight = Game1.anyPlayerTex.Height * 0.35f,
            color = Color.White,
        };
```

- המחלקה של הלקוח במשחק- התחברות לשרת בתוך התהליכון.
- ip= הכתובת של השרת אליו צריך להתחבר.
- port= כלי שבעזרתו מבצעים העברת נתונים באופן ישיר בין כלי תקשורת.
- הפעולה InitPlayers מאתחלת את הדמויות ומגדירה אותן כך: serverPlayer מוגדר כאויב ו- clientPlayer מוגדר כשחקן הנוכחי.

```
/// <summary>
/// finds the server and transports data in a connection.
/// </summary>
protected override void SocketThread()
{
    client = new TcpClient();
    client.Connect(hostip, port);

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

    ConnectionEventTrigger();

    while (true)
    {
        ReadAndUpdateCharacter(serverPlayer);
        WriteCharacterData(clientPlayer);

        Thread.Sleep(50);
    }
}
```

- הפעולה דורסת את זאת של OnlineAbst, ותפקידה להתחבר באמצעות ה- ip וה- port לשרת הספציפי ולהעביר נתונים ביניהם.
- בדומה אצל השרת, גם בפעולה זו קיימת קריאה ל- event שהוגדר במחלקת האם, שברגע החיבור יהיה ניתן לזהותו ברחבי ה- namespace.
- הלולאה האינסופית שומרת על החיבור פעיל עד שייקטע החיבור.
- השורה `Thread.Sleep(50);` גורמת לדיליי מכוון בהעברת הנתונים, כדי להאט את הרצת המשחק בהתאם לנראות על המסך.

המחלקה MultiPlayer:

```
class MultiPlayer : State
{
    enum OnlineState // different states of the game.
    {
        AskingRole, //host or join
        Connecting,
        Playing
    }

    OnlineAbst onlineManager;

    OnlineState state = OnlineState.AskingRole; // the initial state of the game.

    public Dictionary<string, Animation> animations;

    List<Platform> platforms = new List<Platform>();

    KeyboardState _currentKey;
    KeyboardState _previousKey;


    /// <summary>
    /// MultiPlayer's constructor.
    /// </summary>
    public MultiPlayer(Game1 game, GraphicsDevice graphicsDevice, ContentManager content) : base(game, graphicsDevice, content)
    {
        animations = new Dictionary<string, Animation>()...;

        var platTex = content.Load<Texture2D>("BG/snowPlat");

        platforms.Add(new Platform(platTex, new Vector2(20, 300)));
        platforms.Add(new Platform(platTex, new Vector2(20, 500)));
    }
```

- בפעולה זו מנוהל המשחק כאשר נלחץ כפתור ה- Multiplayer בתפריט הראשי.
- במחלקה זו יש שימוש ב- enum- משתנה המחזיק מספר סוגים של אותו משתנה אשר אינם מכילים משתנה מוגדר מראש. ה- enum במחלקה זו מכיל את המצבים השונים של המשחק הרב משתתפים. ה- enum ששמו state מוגדר בהתחלה בתור AskingRole- מצב בו אין חיבור/בקשה לחיבור.
- בפעולה הבונה של Multiplayer, ישנה טעינה של משתנים שחיוניים לתפעול המשחק- האנימציות והפלטפורמות.

```
/// <summary>
/// overrides State's Update func
/// </summary>
public override void Update(GameTime gameTime)
{
    _previousKey = _currentKey;
    _currentKey = Keyboard.GetState();

    switch (state)
    {
        case OnlineState.AskingRole:
            if (Keyboard.GetState().IsKeyDown(Keys.H)) // sets the user as server.
            {
                onlineManager = new ServerComp(5050);
                onlineManager.connectHandler += onlineGame_OnConnection;
                onlineManager.Initialize(animations, _content);

                state = OnlineState.Connecting;
            }
            else if (Keyboard.GetState().IsKeyDown(Keys.J)) // sets the user as client.
            {
                onlineManager = new ClientComp("127.0.0.1", 5050);
                onlineManager.connectHandler += onlineGame_OnConnection;
                onlineManager.Initialize(animations, _content);

                state = OnlineState.Connecting; // initiates connecting.
            }
            break;

        case OnlineState.Connecting:

            break;
```

- בפעולת העדכון של המחלקה, יש שימוש ב- switch- דרך לעבור על enums ולהגדיר את מה שיקרה בכל מצב ומצב.
- במקרה של AskingRole, ישנו פיצול למקרה שבו המשתמש מוגדר כשרת (לחיצה על H), ומקרה בו המשתמש מוגדר כלקוח (לחיצה על J). בכל אחד ממצבים אלה, יש קריאה לפעולות של ServerComp ו- ClientComp בהתאמה, וכל אחד מכלי התקשורת מחפש ליצור חיבור עם הכלי השני.
- במצב של חיבור לא קורה כלום, מכיוון שהוא מצב זמני עד לחיבור עצמו, כאשר המצב יהיה Playing.

```
    case OnlineState.Playing:
        onlineManager.serverPlayer.Update(gameTime, onlineManager.clientPlayer); // updates the player/enemy.
        onlineManager.clientPlayer.Update(gameTime, onlineManager.serverPlayer); // updates the player/enemy.

        foreach (var platform in platforms) // platforms update
        {
            if (onlineManager.serverPlayer.Rectangle.isOnTopOf(platform.rectangle))
            {
                onlineManager.serverPlayer.Velocity.Y = 0f;
                onlineManager.clientPlayer.hasJumped = false;

                if (_currentKey.IsKeyUp(Keys.S) &&
                _previousKey.IsKeyDown(Keys.S))
                {
                    onlineManager.serverPlayer.hasJumped = true;
                    onlineManager.serverPlayer.Velocity.Y += 3f;
                }
            }

            if (onlineManager.serverPlayer._position.Y + onlineManager.serverPlayer.texHeight >= Game1.ScreenHeight)
            {
                onlineManager.serverPlayer.hasJumped = false;
            }



            if (onlineManager.clientPlayer.Rectangle.isOnTopOf(platform.rectangle))
            {
                onlineManager.clientPlayer.Velocity.Y = 0f;
                onlineManager.clientPlayer.hasJumped = false;

                if (_currentKey.IsKeyUp(Keys.S) &&
                _previousKey.IsKeyDown(Keys.S))
                {
                    onlineManager.clientPlayer.hasJumped = true;
                    onlineManager.clientPlayer.Velocity.Y += 3f;
                }
            }

            if (onlineManager.clientPlayer._position.Y + onlineManager.clientPlayer.texHeight / 4 >= Game1.ScreenHeight)
            {
                onlineManager.clientPlayer.hasJumped = false;

            }
        }

        break;
}

_game.Window.Title = state.ToString();// prints the current state
```

- במצב של חיבור בין כלי התקשורת, מתחיל לפעול המשחק: הדמויות מתעדכנות ויש בדיקה אם הן נחתו על הפלטפורמות.
- בפעולה זו מתעדכנת גם כותרת המשחק, שבאמצעותה ניתן להצביע במהלך הריצה על המצב בו הוא נמצא.

```
/// <summary>
/// the func that is called when there is a connection.
/// </summary>
private void onlineGame_OnConnection()
{
    state = OnlineState.Playing;
}

/// <summary>
/// overrides State's Draw func
/// </summary>
public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
    if (state == OnlineState.Playing)
    {
        onlineManager.clientPlayer.Draw(spriteBatch);
        onlineManager.serverPlayer.Draw(spriteBatch);

        foreach (var platform in platforms)
        {
            platform.Draw(spriteBatch);
        }
    }
}
```

- כאשר נוצר החיבור, ישנה קריאה ל- Online_OnConnection שבה יש עדכון של המצב של המשחק מ- "ממתין לחיבור" ל- "נוצר חיבור".
- בפעולה Draw ישנו ציור של הדמויות והפלטפורמות רק כאשר ישנו חיבור והמשחק פעיל.

כיצד להריץ את מצב רב המשתתפים על מחשב אחד?

1. *יש ללחוץ ctrl F5 פעם אחת, שיפתח חלונית אחת של המשחק.*
2. *יש ללחוץ ctrl F5 שוב, כדי שתיפתח חלונית שניה.*
3. *יש ללחוץ בכל אחת מהחלוניות על הכפתור Multiplayer בתפריט.*
4. *יש חיבור, והמשחק מתחיל.*

חלק חמישי- המחלקה המרכזית:

המחלקה Game1:

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

    public static int ScreenWidth;
    public static int ScreenHeight;

    SnowGenerator snow;

    Player player;
    Bot bot;

    private SpriteFont _font; // to write on screen

    //static for using in other classes.
    public static Song _menuSong;
    public static Song _duelSong;
    public static SoundEffect _throwing;
    public static SoundEffect _click;
    public static SoundEffect _hovering;

    SoundEffect victory;
    SoundEffect defeat;
    SoundEffect hit;
    SoundEffect exitClick;

    public static Texture2D anyPlayerTex;

    public static Texture2D knifeTexture;

    public static Texture2D healthTex;
```

- הכרזה על משתנים של אודיו, תמונות והשחקנים במשחק.
- חלק מהמשתנים מוגדרים כ- static, כדי שיהיה ניתן לגשת אליהם במחלקות אחרות בפרויקט. למשל, המשתנה _click הוא סטטי, ובמחלקה אחרת אני יכול לגשת אליו באמצעות Game1._click.

```
Texture2D bg;
Texture2D optionsTex;
Texture2D title;
Texture2D guide;
Texture2D outLine;
Texture2D vicTex;
Texture2D defTex;

KeyboardState _currentKey;
KeyboardState _previousKey;
Button exitButton; // for optionsMenu

List<Platform> platforms = new List<Platform>();

public static bool escPress = false; // to check if esc is pressed once
public static bool activeGame = false;
public static bool inMenu = true; //for returning to game & menu in optionsMenu
bool anywon = false;

int counterVic = 0;
int counterDef = 0;

#region States
private State _currentState;

private State _nextState;

//for optionsMenu
private State _currentOPstate;

private State _nextOPstate;

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

#endregion
```

- עוד הגדרות של משתנים- תמונות במשחק, הרשימה של הפלטפורמות, ניהול המצבים (States) של תפריט הפתיחה ותפריט עצירת המשחק, וכמה משתנים מיוחדים:
- activeGame- מכיל אמת אם המשחק התחיל ורץ, אחרת שקר (במצב של תפריט הפתיחה או תפריט עצירת המשחק).
- counterVic- כדי לוודא שהשחקן/אויב יוכלו לנצח רק פעם אחת, למשתנה זה מתווסף 1 במקרה של ניצחון- השימוש בכך יצוין בהמשך.
- counterDef- אותה מטרה, רק במקרה של ניצחון.
- הפעולה ChangeState מטפלת במצב של שינוי המצב- מעבר מהתפריט הראשי למצב שחקן יחיד או למצב רב משתתפים.

```
/// <summary>
/// loads all the content (Texture2D, Song, SoundEffect, Font) from Content file.
/// </summary>
protected override void LoadContent()
{
    spriteBatch = new SpriteBatch(GraphicsDevice);
    _menuSong = Content.Load<Song>("Audio/intro music");
    MediaPlayer.Play(_menuSong);
    MediaPlayer.Volume = 0.3f;
    MediaPlayer.IsRepeating = true; // infinite loop

    title = Content.Load<Texture2D>("General/GameTitle");
    _throwing = Content.Load<SoundEffect>("Audio/throwing");
    victory = Content.Load<SoundEffect>("Audio/victory");
    defeat = Content.Load<SoundEffect>("Audio/defeat");
    hit = Content.Load<SoundEffect>("Audio/ouch");
    _hovering = Content.Load<SoundEffect>("Audio/hovering");
    _click = Content.Load<SoundEffect>("Audio/button click");
    exitClick = Content.Load<SoundEffect>("Audio/exit click");
    _duelSong = Content.Load<Song>("Audio/Warrior Rising");
    knifeTexture = Content.Load<Texture2D>("Objects/Kunai");
    optionsTex = Content.Load<Texture2D>("General/options frame");
    vicTex = Content.Load<Texture2D>("General/victory");
    defTex = Content.Load<Texture2D>("General/defeat");
    outLine = Content.Load<Texture2D>("Objects/HealthBarOutLine");
    healthTex = Content.Load<Texture2D>("Objects/HealthBarInside");
    anyPlayerTex = Content.Load<Texture2D>("AnimationFrames/runningR");
    _font = Content.Load<SpriteFont>("Fonts/Font");
    Texture2D platTex = Content.Load<Texture2D>("General/snowPlat");
    Texture2D buttonTex = Content.Load<Texture2D>("Controls/blueButton");
    bg = Content.Load<Texture2D>("General/snowBack2");
    guide = Content.Load<Texture2D>("General/Guide");

    snow = new SnowGenerator(Content.Load<Texture2D>("Snow/smallParticle"), ScreenWidth, 100);
```

- בפעולה LoadContent טוענים קבצים מתיקיית content- תמונות, קבצי שמע ופונטים אל המשתנים שהוגדרו לעיל.
- בנוסף, נוצר המשתנה snow מסוג SnowGenerator שמקבל תמונה של כדור שלג אחד, גודל המסך בו הוא ייפול והמהירות בה כדורי השלג ייפלו.

```
Dictionary<string, Animation> animations = new Dictionary<string, Animation>() // gets the animation and num of frames.
{
    {"runningR", new Animation(Content.Load<Texture2D>("AnimationFrames/runningR"), 10)},
    {"runningL", new Animation(Content.Load<Texture2D>("AnimationFrames/runningL"), 10)},
    {"throwingR", new Animation(Content.Load<Texture2D>("AnimationFrames/throwingR"), 10)},
    {"throwingL", new Animation(Content.Load<Texture2D>("AnimationFrames/throwingL"), 10)},
    {"standingL", new Animation(Content.Load<Texture2D>("AnimationFrames/standingL"), 10)},
    {"standingR", new Animation(Content.Load<Texture2D>("AnimationFrames/standingR"), 10)},
    {"jumpingR", new Animation(Content.Load<Texture2D>("AnimationFrames/jumpingR"), 10)},
    {"jumpingL", new Animation(Content.Load<Texture2D>("AnimationFrames/jumpingL"), 10)},
};

player = new Player(animations);

bot = new Bot(animations, player);

platforms.Add(new Platform(platTex, new Vector2(20, 300)));
platforms.Add(new Platform(platTex, new Vector2(20, 500)));

_currentState = new MainMenu(this, graphics.GraphicsDevice, Content);
_currentOPstate = new OptionsMenu(this, graphics.GraphicsDevice, Content);

exitButton = new Button(buttonTex, _font)
{
    Text = "Exit",
    Position = new Vector2(ScreenWidth / 2 - buttonTex.Width / 2, 400) //roughly center of screen
};

exitButton.Click += ExitButton_Click;
```

```
}

/// <summary>
/// when called, exits the game.
/// </summary>
private void ExitButton_Click(object sender, System.EventArgs e)
{
    Exit();
}
```

- המשך של הפעולה, בו מוגדר מילון האנימציות של הדמויות, מוגדרות הדמויות של מצב singlePlayer, מתמלאת רשימת הפלטפורמות, מוגדרים המצבים וכפתור היציאה.
- בנוסף יש הוספה של הפעולה ExitButton_Click ל- event של כפתור היציאה, שכאשר יש קריאה לפעולה זו המשחק נגמר.

```
/// <summary>
/// updates the different elements of the game.
/// </summary>
protected override void Update(GameTime gameTime)
{
    // used to check if a key is pressed once.
    _previousKey = _currentKey;
    _currentKey = Keyboard.GetState();

    snow.Update(gameTime, graphics.GraphicsDevice);

    if (anywon) // if either player or enemy has won.
    {
        exitButton.Update(gameTime);
    }

    if (_currentKey.IsKeyDown(Keys.Escape))
    {
        escPress = true;
    }
    if (_currentOPstate.gameActivation && escPress && !_currentOPstate.returnToMenu) //stops the game and updates the options menu.
    {
        activeGame = false;
        MediaPlayer.Stop();
        if (_nextOPstate != null)
        {
            _currentOPstate = _nextOPstate;

            _nextOPstate = null;
        }

        _currentOPstate.Update(gameTime);

    }
```

- הפעולה Update מעדכנת תחילה את השלג (ניתן להבחין שהעדכון שלו לא נמצא בתנאי כלשהו- משמע השלג ייפול ברקע בלי קשר להתרחשויות במשחק).
- לאחר מכן יש ניהול של תפריט האפשרויות במצב של עצירת משחק- כאשר המשתמש לוחץ על esc, עדכון המשחק נפסק (activeGame = false), המוזיקה מפסיקה (MediaPlayer.Stop()), והתפריט מתעדכן.

```
if (activeGame)
{
    #region platforms

    foreach (var platform in platforms)
    {
        if (player.Rectangle.isOnTopOf(platform.rectangle)) // if the player has landed on a platform.
        {
            player.Velocity.Y = 0f;
            player.hasJumped = false;

            if (_currentKey.IsKeyUp(player.Input.Down) &&
            _previousKey.IsKeyDown(player.Input.Down))
            {
                player.hasJumped = true;
                player.Velocity.Y += 3f;
            }
        }

        if (player._position.Y + player.texHeight >= ScreenHeight)
        {
            player.hasJumped = false;
        }

        if (bot.Rectangle.isOnTopOf(platform.rectangle))// if the bot has landed on a platform.
        {
            bot.Velocity.Y = 0f;
            bot.hasJumped = false;

            if (bot.decision.Equals("down"))
            {
                bot.hasJumped = true;
                bot.Velocity.Y += 3f;
            }
        }

        if (bot._position.Y + bot.texHeight / 4 >= ScreenHeight)
        {
            bot.hasJumped = false;

        }
    }

    #endregion
```

- בתוך תנאי זה, מנוהל העדכון של האלמנטים במשחק עצמו.
- תחילה, ישנה לולאה שעוברת על כל הפלטפורמות ובודקת אם השחקן או הבוט נחתו על אחת מהן.

```
player.Update(gameTime, bot);
bot.Update(gameTime, player);

PostUpdate(); //knife managing.

if (_currentOPstate.returnToMenu)// when going back to menu, the game resets.
{
    _currentState.gameActivation = false;
    resetPlayers();
    _currentState.guideActivation = false;
    _currentOPstate.returnToMenu = false;
    inMenu = true;

}
if (!_currentState.guideActivation && !_currentState.gameActivation)
{
    _currentState.Update(gameTime);
    inMenu = true;
}
```

- בהמשך הפעולה, ישנו עדכון של הדמויות (קריאה לפעולה Update עליה פורט במחלקה Sprite), וטיפול במצב של חזרה לתפריט הראשי דרך תפריט עצירת המשחק. אם נלחץ הכפתור Main Menu, אז:
- המשתנים הבוליאניים שאחראיים להפעלת המשחק יהיו שווים ל- false.
- קריאה לפעולה שמאפסת את מד החיים של השחקנים ומחזירה אותם למיקומים ההתחלתי.
- ישנו עוד תנאי של עדכון התפריט הראשי, כדי לכסות מצב בו השחקן מתחיל את המשחק וחוזר לתפריט הראשי דרך תפריט עצירת המשחק.

```
    #region States updates
    if (_nextState != null)
    {
        _currentState = _nextState;

        _nextState = null;
    }

    if (!_currentState.guideActivation && !_currentState.gameActivation) // in main menu
    {
        _currentState.Update(gameTime);
    }

    #endregion

    #region win&lose scenario
    if (player.hasWon && counterVic == 0)// enabling the player to win once before reseting.
    {
        anywon = true;
        MediaPlayer.Stop();
        victory.Play(volume: 0.2f, pitch: 0.0f, pan: 0.0f);
        counterVic++;
    }
    if (bot.hasWon && counterDef == 0)// enabling the player to win once before reseting.
    {
        counterDef++;
        anywon = true;
        MediaPlayer.Stop();
        defeat.Play(volume: 0.2f, pitch: 0.0f, pan: 0.0f);
    }
    #endregion

    base.Update(gameTime);
}
```

- עדכון של הכפתורים בתפריט הפתיחה.
- טיפול במצב של ניצחון של השחקן- המשחק נעצר, המוזיקה מפסיקה, ומופעל סאונד של ניצחון.
- טיפול במצב של ניצחון של הבוט- המשחק נעצר, המוזיקה מפסיקה ומופעל סאונד של הפסד.

```
/// <summary>
/// removes the knife from set if one hit the player/enemy.
/// </summary>
private void PostUpdate()
{
    for (int i = 0; i < player.knivesSet.Count; i++)
    {
        if (player.knivesSet[i].IsRemoved)
        {
            hit.Play(volume: 0.15f, pitch: 0.0f, pan: 0.0f);
            player.knivesSet.RemoveAt(i);
            i--;
        }
    }

    for (int i = 0; i < bot.knivesSet.Count; i++)
    {
        if (bot.knivesSet[i].IsRemoved)
        {
            hit.Play(volume: 0.15f, pitch: 0.0f, pan: 0.0f);
            bot.knivesSet.RemoveAt(i);
            i--;
        }
    }
}

/// <summary>
/// max their health back and takes them back to their positions.
/// </summary>
private void resetPlayers()
{
    player.health.rectangle.Width = Content.Load<Texture2D>("Objects/HealthBarInside").Width - 20;
    player._position.X = 50;
    player._position.Y = 700;
    player.knivesSet.Clear();

    bot.health.rectangle.Width = Content.Load<Texture2D>("Objects/HealthBarInside").Width - 20;
    bot._position.X = 1200;
    bot._position.Y = 700;
    bot.knivesSet.Clear();

}
```

- שתי פעולות עזר:
- PostUpdate –בודק האם סכין של השחקן פגעה בבוט ולהפך, ובמקרה שזה קרה יהיה סאונד של פגיעה, והסכין תיעלם מהמסך ומסט הסכינים של הדמות.
- ResetPlayers- פעולה אליה יש קריאה במצב של חזרה לתפריט הראשי. הפעולה מאפסת את השחקנים- מד החיים שלהם מתמלא, הם חוזרים למיקום ההתחלתי וסט הסכינים שלהם מתרוקן.

```
/// <summary>
/// draws the entire game.
/// </summary>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    spriteBatch.Begin();

    spriteBatch.Draw(bg, new Vector2(0, 0), Color.White);
    snow.Draw(spriteBatch);

    if (!_currentState.gameActivation)//if in main menu
    {
        spriteBatch.Draw(title, new Vector2(350, 30), null, Color.White, 0, Vector2.Zero, 0.5f, SpriteEffects.None, 1);

        _currentState.Draw(gameTime, spriteBatch); // drawing the menu

        #region guide
        if (_currentState.guideActivation == true)
        {
            spriteBatch.Draw(guide, new Vector2(225, 125), null, Color.White, 0f, Vector2.Zero, 0.6f, SpriteEffects.None, 1);
            if (_currentKey.IsKeyUp(Keys.Escape) && _previousKey.IsKeyDown(Keys.Escape))
            {
                exitClick.Play(volume: 0.3f, pitch: 0.0f, pan: 0.0f);
                _currentState.guideActivation = false;
            }
        }
        #endregion

    }
```

- הפעולה Draw מציירת על המסך את כל הדמויות, עצמים ואלמנטים של המשחק, באמצעות שימוש בכלי SpriteBatch- "מכחול" שמקבל תמונה/טקסט ומשתנים נוספים ומציג אותם על המסך. גם הפעולה Draw וגם הפעולה Update נקראות 60 פעמים בשנייה, כך שלמשל הצגת אנימציה או תזוזה של אובייקט תהיה חלקה לאור העובדה שהמסך נמחק ומצויר מחדש 60 פעמים בשנייה.
- בפעולה עצמה, תחילה מצויר הרקע והשלג- שאינם תלויים בשום תנאי, כלומר יצוירו בכל שלב של המשחק.
- לאחר מכן, מצויר המסך במצב של תפריט הפתיחה- מצוירת הכותרת והכפתורים של התפריט.
- אם המשתמש לחץ על הכפתור ?How to Play, יצויר המדריך של כיצד משחקים ורקע למשחק.
- בנוסף אם המשתמש יוצא מהמדריך יהיה סאונד מיוחד שיושמע.

```
else
{
    #region drawing the game
    _currentOPstate.gameActivation = true;
    spriteBatch.Draw(outLine, new Vector2(115, 94), null, Color.White, 0, Vector2.Zero, 0.4f, SpriteEffects.None, 1);
    spriteBatch.Draw(outLine, new Vector2(1015, 94), null, Color.White, 0, Vector2.Zero, 0.4f, SpriteEffects.None, 1);

    spriteBatch.DrawString(_font, "My Player", new Vector2(150, 50), Color.Black);
    spriteBatch.DrawString(_font, "Enemy", new Vector2(1050, 50), Color.Black);

    foreach (var platform in platforms)
    {
        platform.Draw(spriteBatch);
    }

    player.Draw(spriteBatch);
    bot.Draw(spriteBatch);

    if (_currentOPstate.gameActivation && escPress)// to draw the options menu
    {
        spriteBatch.Draw(optionsTex, new Vector2(390, 60), null, Color.White, 0f, Vector2.Zero, 0.9f, SpriteEffects.None, 1);
        _currentOPstate.Draw(gameTime, spriteBatch);
    }


    #region win&lose scenario
    if (player.hasWon)
    {
        activeGame = false;
        var vicX = ScreenWidth / 2 - vicTex.Width / 2 - 5;
        spriteBatch.Draw(vicTex, new Vector2(vicX, 100), Color.White);
        exitButton.Draw(gameTime, spriteBatch);
    }

    if (bot.hasWon)
    {
        activeGame = false;
        spriteBatch.Draw(defTex, new Vector2((ScreenWidth / 2 - defTex.Width / 2) + 5, 100), Color.White);
        exitButton.Draw(gameTime, spriteBatch);
    }
    #endregion


    #endregion
}

spriteBatch.End();

base.Draw(gameTime);
```

- בקוד זה, מצויר המשחק עצמו:
- מצוירים מדי החיים של השחקנים, הפלטפורמות והשחקנים.
- במצב של לחיצה על esc, יצויר תפריט עצירת המשחק.
- במצב של ניצחון או הפסד של השחקן, המשחק ייעצר, יושמע סאונד מיוחד, תצויר תמונה של ניצחון/הפסד ויופיע כפתור של יציאה מהמשחק.

```
// a class that helps with the platforms.
static class RectangleHelper
{
    const int penetrationMargin = 47; // custom number chosen by viewing the game.

    /// <summary>
    /// checks if the character has landed on the platform.
    /// </summary>
    /// <param name="r1"> character's rectangle</param>
    /// <param name="r2"> platform's rectangle</param>
    ///
    public static bool isOnTopOf(this Rectangle r1, Rectangle r2)
    {
        return r1.Bottom >= r2.Top + penetrationMargin &&
                r1.Bottom <= r2.Top + (r2.Height / 2 + 32);
    }

}
```

- המחלקה הסטטית RectangleHelper, שבה נבדק היחס בין מלבן של השחקן לבין המלבן של הפלטפורמה- המחלקה מכילה את הפעולה הסטטית isOnTopOf, שמחזירה אמת אם המלבן של השחקן נמצא על המלבן של הפלטפורמה, אחרת שקר.
- פעולה זו זמינה לכל מחלקה ב- namespace, ומשומשת בפעולה Update בה נבדק האם שחקן כלשהו נחת על פלטפורמה או לא.

רפלקציה- בעיות פתוחות והצעות לשיפור:

כפי שתיארתי בהקדמה של חלק הקוד בתיק, החלטתי לחלק את העבודה שלי לחלקים הגיוניים שיקלו על התקדמות יעילה וימנעו בעיות רציניות במבנה הקוד בהמשך. החלטתי ליצור משחק ראשוני תחילה, שהאלמנטים הבסיסים שבו יעבדו, ואחר כך לשכלל ולשכלל אותו עד שיגיע למצב סופי ומספק. היו לא מעט אתגרים בתהליך זה- נעזרתי מדי פעם באינטרנט בשביל אלגוריתמים מסוימים, ותהליך המיזוג של אלגוריתמים אלה בקוד הנוכחי שלי היה מסובך לעתים ומאתגר. ככל שהפרויקט משתכלל ונוספים עוד ועוד אלמנטים, כך נוצר יותר קושי להוסיף קטעים חדשים ללא להיפגע מטעויות ובאגים. בתחילת התהליך גיליתי שככל שאני אשקיע יותר זמן והשקעה בחלק מסוים, כך אוכל לצלוח אותו ולגרום לקוד לעבוד כראוי, לכן דרך הפתרון שלי לרוב הבעיות הייתה השקעה של זמן ומאמץ עד שהבעיה תיפתר.

זהו הפרויקט השני שהכנתי, ומפרויקט לפרויקט אני מגלה על עצמי יכולות פתירת בעיות והתגברות על קשיים שמתפתחות תוך עבודה קשה ועקשנות. בנוסף, נוכחתי לדעת שהיכולת לעשות debug לקוד מאוד שימושית, ומצאתי את עצמי משתמש באפשרות זו פעמים רבות כדי לפתור בעיות. הלקח שאני אקח להכנת הפרויקטים הבאים הוא ניסיון לשמור על עבודה רציפה על העבודה- לאור המקצועות האחרים בהם הייתי צריך ללמוד, תהליך העבודה שלי על הפרויקט השתרך על כמעט שבעה חודשים, עם הפסקות ארוכות לצורך מקצועות אחרים. להבא אני אקפיד על תקופות מסודרות בהן אעבוד ברציפות וכתוצאה מכך ביעילות ובמהירות.

לצערי, למרות עבודה ממושכת הפרויקט שלי אינו מושלם, ולוקה במספר בעיות קטנות: למרות שהקוד עובד, דרך הפתרון שלי לחלק מהבעיות בניהול ה- States מעט בעייתית- אני משתמש במספר משתנים שמנהלים את המצבים. הדרך האידיאלית יותר לפתור את בעיה זו היא לפצל את ניהול המצבים למחלקות נפרדות- לכל מצב תהיה מחלקה ובה הוא יהיה מנוהל. במצב רב המשתתפים אני פועל כך, אך בשאר המצבים אני משתמש בדרך לא אידיאלית שפוגעת בזרימה של הקוד, למרות שהיא עובדת. בעיה נוספת שאינה פוגעת באיכות המשחק עצמו אך מזיקה לאלגנטיות של הקוד היא הפשטות של הבינה המלאכותית- יצרתי אלגוריתם של הבוט שעובד וניתן לשנותו לפי דרגת קושי והעדפה אישית, אך הייתי יכול להפוך אותו למורכב יותר.

נספחים- שאר הקוד בפרויקט:

המחלקה Input:

```
public class Input
{
    public Keys Down { get; set; }

    public Keys Up { get; set; }

    public Keys Right { get; set; }

    public Keys Left { get; set; }

    public Keys Throw { get; set; }

    public Keys Jump { get; set; }
}
```

המחלקה Platform:

```
class Platform : Sprite
{

    /// <summary>
    /// Platform's constructor.
    /// </summary>
    public Platform(Texture2D newTexture) : base(newTexture)
    {
    }

    /// <summary>
    /// overrides Sprite's Draw function
    /// </summary>
    public override void Draw(SpriteBatch spriteBatch)
    {
        spriteBatch.Draw(_texture, rec, Color.White);
    }
}
```

המחלקה Snowfall:

```
class Snowfall
{
    Texture2D texture;
    Vector2 position;
    Vector2 velocity;

    public Vector2 Position
    {
        get { return position; }
    }

    /// <summary>
    /// Snowfall's constructor.
    /// </summary>
    public Snowfall(Texture2D newTexture, Vector2 newPosition, Vector2 newVelocity)
    {
        texture = newTexture;
        position = newPosition;
        velocity = newVelocity;
    }

    /// <summary>
    /// updates the position.
    /// </summary>
    public void Update()
    {
        position += velocity;
    }

    /// <summary>
    /// draws the snowfall.
    /// </summary>
    public void Draw(SpriteBatch sb)
    {
        sb.Draw(texture, position, Color.White);
    }
}
```

המחלקה SnowGenerator:

```
class SnowGenerator
{
    Texture2D texture;

    float spawnWidth;
    float density;

    List<Snowfall> snowFall = new List<Snowfall>();

    float timer;

    Random rand1, rand2;

    /// <summary>
    /// SnowGenerator's constructor.
    /// </summary>
    public SnowGenerator(Texture2D newTexture, float newSpawnWidth, float newDensity)
    {
        texture = newTexture;
        spawnWidth = newSpawnWidth;
        density = newDensity;
        rand1 = new Random();
        rand2 = new Random();
    }
```

```
/// <summary>
/// adds to the list a snow particle.
/// </summary>
public void CreateSnowParticle()
{
    snowFall.Add(new Snowfall(texture, new Vector2(
        -50 + (float)rand1.NextDouble() * spawnWidth, 0),
        new Vector2(1, rand2.Next(5, 8))));
}

/// <summary>
/// updates the positions of the snowfall and
/// delete the ones who reach the bottom of screen
/// </summary>
public void Update(GameTime gameTime, GraphicsDevice graphics)
{
    timer += (float)gameTime.ElapsedGameTime.TotalSeconds;

    while (timer > 0)
    {
        timer -= 1f / density;
        CreateSnowParticle();
    }
    for (int i = 0; i < snowFall.Count; i++)
    {
        snowFall[i].Update();

        if (snowFall[i].Position.Y > graphics.Viewport.Height)
        {
            snowFall.RemoveAt(i);
            i--;
        }
    }
}

/// <summary>
/// draws the snowfall.
/// </summary>
public void Draw(SpriteBatch sb)
{
    foreach (Snowfall snow in snowFall)
        snow.Draw(sb);
}
```

המחלקה Component:

```
public abstract class Component
{
    /// <summary>
    /// basic update function.
    /// </summary>
    public abstract void Update(GameTime gameTime);

    /// <summary>
    /// basic draw function.
    /// </summary>
    public abstract void Draw(GameTime gameTime, SpriteBatch spriteBatch);

}
```