מדינת ישראל
משרד החינוך
STATE OF ISRAEL
MINISTRY OF EDUCATION

גבהים
חינוך • מצוינות • סייבר

# SMARTICAR

## CAN YOU BEAT THE AI?


**שם המגיש** עוז אלתגר

**ת.ז** 212781249

**שם המנחה** גולן מור


**מאי 2020**

מדינת ישראל
משרד החינוך
STATE OF ISRAEL
MINISTRY OF EDUCATION

**תוכן עניינים**

**תקציר ורציונל הפרוייקט**

כאשר אנו שומעים את צמד המילים "רכב אוטונומי", עולות לראשנו תמונות סרטי מדע בדיוני בהן החברה מנוהלת ע"י רובוטים השולטים בחיינו ובן היתר גם מסיעים אותנו ממקום למקום.
לעומת זאת, לרוב, אנו לא באמת מבינים את עוצמת הדבר. SmartiCar באה לפתור זאת.
SmartiCar הינה מערכת הבנויה בצורת משחק, ומטרתה המחשת יכולותיו של הרכב האוטונומי.
על מנת להמחיש יכולות אלה, המערכת מספקת שני "מודים" בהם המשתמש יכול לנסות את מזלו ולהוכיח שכישוריו עולים על אלה של יריבו המחשב- מסלול מכשולים, בו על המשתמש לסיים שתי הקפות שלמות סביב מסלול המרוצים ללא פגיעה באף מכשול, ומוד בו המנצח הינו הנהג אשר הצליח לחנות בחניון יותר פעמים מיריבו בזמן של כ-60 שניות.

<u>סיבות לבחירת הנושא לפרוייקט</u>

- בשנים האחרונות, תחום הבינה המלאכותית ובפרט תחום הרכבים האוטונומיים צבר תאוצה וחברות רכב משקיעות סכומי כסף ומשאבים רבים בתהליכי הפיתוח והייצור.

- בניית המערכת נעשתה בתכנת Unity, תכנה שהשימוש בה כבר עניין אותי זמן רב כך שפיתוח הפרוייקט הווה הזדמנות מצויינת "ללכלך את הידיים" ולצבור נסיון בעבודה איתה.

- בינה מלאכותית תמיד הצטיירה אצלי "כמדע בדיוני" וכדבר שלא באמת הבנתי איך הוא עובד, אך לאחר העיסוק בחומר הלימוד הבסיסי במהלך השנתיים האחרונות, היה בידי את הבסיס ללמידה עצמית ויכולת ללמוד עוד במטרה להבין לעומק כיצד הדברים עובדים.

**מבוא ורקע כללי**

בינה מלאכותית (AI), היא ענף של מדעי המחשב העוסק ביכולת לתכנת מחשבים לפעול באופן המציג יכולות שאפיינו עד כה את הבינה האנושית בלבד. כיום, בינה מלאכותית מופיעה בחיינו כמעט בכל מקום אפשרי- היא משלימה לנו את החיפושים בגוגל, מציעה לנו שירים על סמך שירים ששמענו בעבר ועוד. תת-תחום של בינה מלאכותית, הינו תחום למידת המכונה (machine learning) אשר עוסק בפיתוח אלגוריתמים המיועדים לאפשר למחשב ללמוד מתוך דוגמאות, ופועל במגוון משימות חישוביות בהן התכנות הקלאסי אינו אפשרי.

נהוג לחלק את אלגוריתמי למידת המכונה למספר סוגים:

- למידה מונחית (supervised learning). כל דוגמה מגיעה ביחד עם תווית סיווג. מטרת האלגוריתם היא לחזות את הסיווג של דוגמאות חדשות שאותן לא פגש בתהליך הלמידה.

- למידה בלתי מונחית (unsupervised learning). מטרת האלגוריתמים היא למצוא ייצוג פשוט וקל להבנה של אוסף הנתונים.

- למידת חיזוק (reinforcement learning). אלגוריתם הלמידה מקבל משוב חלקי על ביצועיו (רק לאחר סיום ביצוע המטלה) ועליו להסיק אילו מהחלטותיו הביאו להצלחה/כישלון.
*במסגרת הפרוייקט, זהו סוג האלגוריתם בו נעשה הלימוד.

מכונית אוטונומית היא מכונית המנווטת ומתגברת על מכשולים ללא התערבות אקטיבית של נהג אנושי ובלא צורך בתכנון מיוחד של הכביש וסביבתו. המכונית מצוידת בחיישנים שונים המעניקים לה אפשרות לחוש את הסביבה כך שכלל המידע אמור לאפשר למכונית האוטונומית לתכנן את המסלול בזמן אמת ולאפשר לה לנסוע בבטחה. תהליך עיבוד המידע ותכנון המסלול אינו היה מתאפשר אלמלא יכולותיהן הרבות של הבינה המלאכותית ולמידת המכונה, כך ששימוש במכונית אוטונומית לצורך הדגמת יכולות הבינה המלאכותית הינו מתבקש.

**מטרת הפרוייקט**

תפקידו של המוצר המוגמר

מטרת המערכת הינה המחשת יכולותיו של הרכב האוטונומי (באופן מצומצם, ככל הניתן).
SmartiCar בנויה בצורת משחק, בעל שני מצבים בהם המשתמש יכול לנסות את מזלו ולהתמודד מול רכב ללא נהג הנשלט בידי המחשב- מסלול מכשולים, בו על המשתמש לסיים שתי הקפות שלמות סביב מסלול המרוצים ללא פגיעה באף מכשול, ומצב שני בו המנצח הינו הנהג אשר הצליח לחנות בחניון יותר פעמים מיריבו בזמן של כ-60 שניות.

דרישות מרכזיות

דרישות פונקציונאליות-

- המערכת תאפשר למשתמש את בחירת "המוד" בו הוא רוצה להתחרות מול המחשב.
- המערכת תפעל לפי חוקי הפיזיקה, לצורך הדגשת היכולות המציאותיות של הרכב.
- המערכת תדע לקבוע באופן ברור מנצח לתחרות מסויימת.
- ממשק המשתמש הגרפי ידע לתפקד בהתאם לפקודותיו של המשתמש.

דרישות לא פונקציונליות-

- על המחשב לנצח ברוב הפעמים על מנת שהמשתמש יבין את רמתו הגבוהה.
- עבודה "חלקה" של המערכת, ללא תקיעות לעתים תכופות.
- אסתטיקת המערכת- על המערכת להראות קלילה ונוחה למשתמש.

<u>תרחישים במערכת</u>

1. לאחר שהמשתמש הפעיל את התכנה, ע"י לחיצה על- SmartiCar.exe, יפתח בפניו מסך המערכת הראשי ובו כפתור START להתחלת השימוש אשר יעביר אותו למסך בחירת המוד.
2. כעת נמצא המשתמש בבחירת מסך המוד בו מופיעות לו שתי אופציות- בחירת מוד מסלול המכשולים (racetrack mode), או בחירת תחרות החנייה (parking mode).

<u>אופציה 1: המשתמש בחר במוד מסלול המכשולים</u>

1. לאחר מסך המראה את תהליך טעינת המוד, המשתמש יועבר למסך בו מוצגות לו הוראות המשחק וכפתור START להתחלתו.
2. לאחר שהמשתמש לחץ על כפתור ה-START, תתחיל ספירה לאחור של כ-3 שניות ומיד עם סיומה יתחיל המשחק.
3. המשתמש התחיל לשחק. כעת, המשחק יגמר ברגע שאחד מהנהגים יסיים 2 סיבובים רצופים ללא התנגשויות בקיר/ מכשול. * למשתמש ישנה אופציה לעצור את המשחק ע"י לחיצה על המקש ESC.
4. ברגע שאחד מן הנהגים ניצח, המשחק יגמר ולמשתמש יוצגו סטטיסטיקות המשחק (מספר הסיבובים שכל אחד מן הנהגים השלים והזמן הכולל של המרוץ). כמו כן, מוצג לרשותו כפתור QUIT ליציאה מן התכנית וסגירת המערכת.

<u>אופציה 2: המשתמש בחר במוד תחרות החנייה</u>

1. לאחר מסך המראה את תהליך טעינת המוד, המשתמש יועבר למסך בו מוצגות לו הוראות המשחק וכפתור START להתחלתו.
2. לאחר שהמשתמש לחץ על כפתור ה-START, תתחיל ספירה לאחור של כ-3 שניות ומיד עם סיומה יתחיל המשחק.
3. המשתמש התחיל לשחק. כעת, המשחק יגמר לאחר 60 שניות.
   * למשתמש ישנה אופציה לעצור את המשחק ע"י לחיצה על המקש ESC.
4. ברגע שנגמר הטיימר, המשחק יגמר ולמשתמש יוצג מספר החניות שכל נהג ביצע.
   כמו כן, מוצג לרשותו כפתור QUIT ליציאה מן התכנית וסגירת המערכת.

**שפת התכנות וסביבת העבודה**

הפיתוח התבצע בעזרת התכנה Unity, אשר הכתיבה בה נעשית בשפת #C.
בנוסף, על מנת להכניס אלמנטים של machine learning לתוך Unity, יש להשתמש בספריית העזר Unity ML-Agents, שמימושה בפייתון- אך לא דורשת כתיבת קוד בפייתון, אלא מספקת API לצורך שימוש בפונקציות הנדרשות.

Unity הוא מנוע גרפי חוצה פלטפורמות שפותח על ידי יוניטי טכנולוגיות בשנת 2005 ומשמש לפיתוח משחקי וידאו למחשב, קונסולות, טלפונים חכמים ואתרי אינטרנט.

שפת #C היא שפת תכנות הפותחה ע"י "מיקרוסופט" בשנת 2000 כחלק מפרויקט "דוט נט" (.NET) ונחשבת לאחת משפות התכנות הפופולריות בעולם. התחביר והעקרונות שלה פשוטים, אך מצד שני עשירים ביכולת.

Python היא שפת תכנות דינמית מהנפוצות ביותר אשר פותחה בשנת 1991 ע"י גואידו ואן רוסום כאשר שמה ניתן לה כמחווה לקבוצה הקומית "מונטי פייתון". פייתון תוכננה תוך שימת דגש על קריאות הקוד, וכוללת מבנים המיועדים לאפשר ביטוי של תוכניות מורכבות בדרך קצרה וברורה.

בחירתי בעבודה עם Unity נבעה מכך שבניית משחקים מורכבים, ברמה שהתכנה מאפשרת, הוא נושא שעניין אותי כבר זמן רב, כך שהפרוייקט הווה הזדמנות מצויינת להתנסות איתה. כמו כן, העבודה ב- Unity נעשית בשפת #C אשר בה אני שולט היטב עקב הלימודים הרבים שנעשו בשנתיים האחרונות, במסגרת הבגרות במדעי המחשב.

**ניסוח וניתוח הבעיה האלגוריתמית**

במהלך העבודה, נתקלתי במספר בעיות אלגוריתמיות-

1. ראשית, הבעיה האלגוריתמית המרכזית בפרוייקט- גרימה לכך שהמחשב יוכל לנהוג בכוחות עצמו בצורה המיטבית. על מנת לעשות זאת, היה עליי לקחת בחשבון כמה דברים עיקריים-
   - מהו ה-input אשר יהיה עליי לספק למחשב כך שיכיל את הנתונים החשובים ביותר על מנת לקבל החלטות ולפעול על פיהן, ללא העמסת מידע שעלולה להאט את תהליך הלימוד.
   - מהו ה- output שארצה לקבל מהמחשב, כלומר- איזה פעולות הרכב יוכל לקחת על סמך הנתונים שקיבל.
   - בחירת סוג אלגוריתם ה-machine learning שבו ארצה ללמד את המחשב.

2. שנית, גרימה לכך שהמשחק יכיל אלמנטים פיזיקליים ראליסטים, כך שיורגש הפן המציאותי. לשם כך, יש לשים לב לדגשים כגון-
   - תאוצת הרכב ושליטה בהיגוי.
   - טיפול בהתנגשויות הרכב עם גורמים שונים (מכשולים, קירות וכ'ו).
   - פרופורציות בין גדלי גופים שונים בסביבה כגון- בין הרכב לכביש, בין עצים ליציע וכ'ו.

3. לבסוף, שליטה בחוקי המשחק וטיפול בעברות/פסילות-
   - אפשרות לעצור את מעבר הזמן במהלך מסך הוראות/עצור/סיום.
   - אתחול הרכב לאחר התנגשות.
   - ספירת התוצאות ומעקב אחרי זמנים.

**תיאור אלגוריתמים קיימים**

<u>פתרון הבעיה הראשונה</u>

לאחר מחקר מעמיק, בו הבנתי לעומק מה ההבדל בין כל סוג machine learning, כיצד הדברים עובדים בפועל ובאיזה כלים עליי להשתמש, הגעתי למסקנות הבאות-

ראשית, בחירת "משפחת" אלגוריתם ה-machine learning-

בלמידה מונחית (supervised learning), כל דוגמה מגיעה ביחד עם תווית סיווג. מטרת האלגוריתם היא לחזות את הסיווג של דוגמאות חדשות שאותן לא פגש בתהליך הלמידה. כלומר, יהיה עליי לנהוג במשך שעות, לאסוף מידע ולאחר מכן לתת אותו למחשב כך שידע מה נכון לעשות בסיטואציות שונות, ע"פ ההחלטות שקיבלתי. ניתן לראות כי סוג זה של למידה מנוגד לרעיון הפרוייקט! הפרוייקט מסתמך על כך שיכולותיו של הרכב יעלו על יכולותיו של הנהג האנושי ולכן לא ניתן ללמד אותו בדרך שבה לבסוף רמתו תשתווה לרמתו של הנהג. לעומת זאת, ניתן להתבונן בשיטת למידת החיזוק ( reinforcement learning) - אלגוריתם הלמידה מקבל משוב חלקי על ביצועיו (רק לאחר סיום ביצוע המטלה) ועליו להסיק אילו מהחלטותיו הביאו להצלחה/כישלון. בדרך זו, הרכב ינהג, יאסוף מידע מהסביבה, ייקח החלטות (בהתחלה כנראה ויהיו שגויות) כך שלאחר כל החלטה שלקח, יקבל משוב חיובי/שלילי בהתאם לתוצאה של אותה החלטה עד שלבסוף יוכל למפות את המידע שיאסוף להחלטות שהביאו לו את המספר הרב ביותר של חיזוקים חיובים.

שנית, על מנת שהמחשב יוכל לקבל את ההחלטות המיטביות, יצטרך להיות בידו את המידע אשר נותן לו תמונה כוללת ככל הניתן של הסביבה בה הוא נמצא, אך עם זאת יש לשים לב כי כמות רבה של מידע עלולה להאט את תהליך הלימוד בצורה משמעותית ואף לגרום לכך שהמחשב יקבל החלטות לא נכונות. המידע הספציפי שהמחשב מקבל משתנה בין שני המודים (מוד מסלול המכשולים ומוד החנייה), אך בכל זאת, לאחר המחקר שערכתי, גיליתי כי הדבר המשותף לשניהם היא העובדה שהרכב יצטרך "לירות" קרני לייזר סביבו (RayCasts) אשר נותנות לו מידע לגבי מרחק המכשולים הנמצאים סביבו ממנו.

לבסוף, מצאתי כי דרך נוספת להקטין משמעותית את זמן הלימוד היא ע"י הגבלת מספר הפעולות שיוכל המחשב לקחת, אך בכל זאת הגבלת הפעולות שיכול לקחת משמע פחות שליטה על הרכב ולכן הגעתי למסקנה שהפעולה הבסיסית שיוכל הרכב לשלוט בו המשותפת לשני המודים, היא היגוי. שליטה במהירות הרכב ועצירתו משתנה בין מוד מסלול המכשולים לבין מוד החנייה.

פתרון הבעיה השנייה

כמו שנאמר, פיתוח המערכת נעשה בעזרת תכנת Unity, ולכן פיתוח ריאליסטי מאוד נח מכוון שהתכנה מספקת כלים שונים לטיפול בנושאים כגון כבידה, התנגשויות ותאוצה/מהירות של גופים.

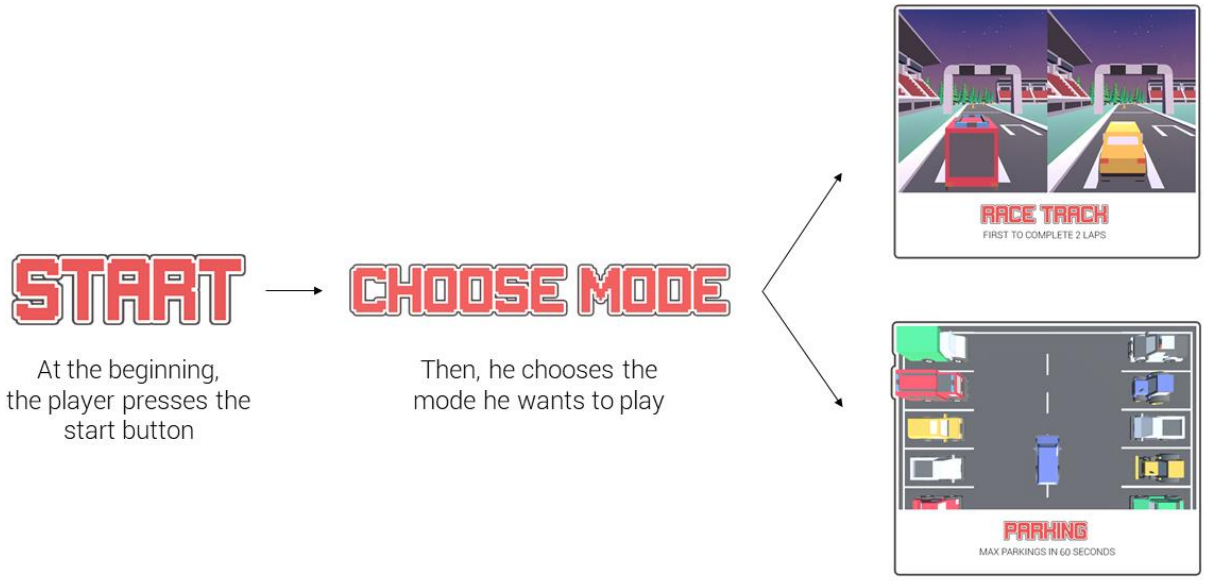
פתרון הבעיה השלישית

בדומה לפתרון הבעיה השנייה, Unity מספקת כלים לעבודה נוחה והתמודדות עם הבעיות שצוינו. על מנת לטפל בנושא הזמנים והתוצאות, עבור כל אחד מהמודים ישנה מחלקה אשר נקראת Manager ותפקידה לנהל את המשחק ע"י שמירת הנתונים הרלוונטיים לכך.

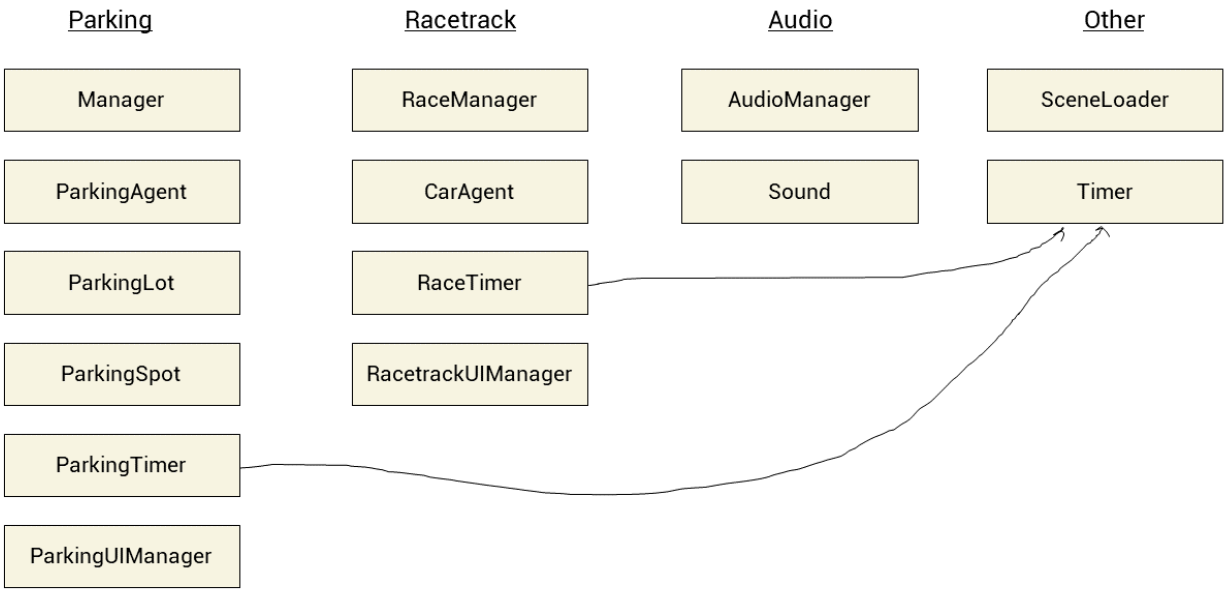
**הפתרון הנבחר**

לאחר הבנת הבעיות האלגוריתמיות לעומק, ניתן היה להסיק כי Unity מספקת כלים נוחים לפתירת שתי הבעיות האחרונות די בקלות, כך שרוב המחקר עסק בפתירת בעיה 1. מרגע שגיליתי כי סוג האלגוריתם המתאים ביותר למטרותיי הינו reinforcement learning, התחלתי בחיפושים אחר כלי שיעזור לי לממש זאת, ולאחר זמן מה, מצאתי את Unity ML-Agents. Unity ML-Agents הינה ספריה אשר נותנת לאמן את המחשב בצורת reinforcement learning ע"י כך שמספקת מימוש ל- neural network ב-python (בעזרת tensorflow) ו-API לתקשר איתה. התקשורת בין השניים מתבצעת ע"י מימוש פונקציה בה מתבצע איסוף המידע הרצוי (CollectObservations()) אשר מוכנס כ-input אל תוך ה-neural network ולאחר מכן מימוש הפונקציה בה ממופה output ה-neural network לפעולות אותן לבסוף הרכב מבצע (OnActionReceived()). ע"י פעולות מתמטיות מורכבות וקובץ בו מופיעים פרמטרים שונים לגבי התנהגותה של ה-neural network (trainer_config.YAML), לבסוף לומד המחשב כיצד לגרום לכמות מרבית של חיזוקים חיוביים ע"י מיפוי המידע שאסף על הסביבה לפעולות אותן יצטרך לקחת. כתוצאה מכך, לאחר ניסויים רבים, הסקתי כי המידע הנחוץ למוד החנייה על מנת לקבל את התוצאה האפקטיבית ביותר הינו (בנוסף ל-Raycasts) מרחק הרכב מהחנייה, הכוון ממנו אל החנייה והכוון שכרגע פונה אליו, ואילו במוד מסלול המכשולים Raycasts הספיקו. כמו כן, במוד החנייה לרכב צריכה להיות את היכולת לנוע בארבעת הכוונים (קדימה, אחורה, ימינה ושמאלה), ואילו במוד מסלול המכשולים לנוע קדימה באופן אוטומטי אך בעל מסוגלות לשלוט בהיגוי.

## מבנה המערכת




## מבנה מחלקות



**פיתוח הפתרון בשכלול הקוד עם שפת התכנות**

ראשית, איסוף המידע אותו מבצע הרכב למען קבלת ההחלטות.

כאשר ישנו שימוש בספריית ML-Agents, איסוף המידע והכנסתו כ-input אל תוך ה-neural network, יש לממש את הפונקציה ()CollectObservations של המחלקה הבסיסית Agent.

מוד החנייה (Parking Mode)

כפי שנאמר בפרק "הפתרון הנבחר", הנתונים אותם אוסף הרכב הינם מרחק הרכב מהחנייה, הכוון ממנו אל החנייה והכוון שכרגע פונה אליו-

```
/// <summary>
/// Collects the agent vector observations in order to feed the neural network policy.
/// The agent calculate it's distance from the parking spot, the direction to the parking spot,
/// and the direction the car is facing.
/// </summary>
/// <param name="sensor">A <see cref="VectorSensor"/> object for adding observations.</param>
3 references
public override void CollectObservations(VectorSensor sensor)
{
    Vector3 parkingSpotPosition = parkingSpot.transform.position;


    // The distance from the parking spot
    sensor.AddObservation(Vector3.Distance(transform.position, parkingSpotPosition));

    // The direction to the parking spot
    sensor.AddObservation((parkingSpotPosition - transform.position).normalized);

    // The direction the car is facing
    sensor.AddObservation(transform.forward);
}
```

מוד מסלול המכשולים (Racetrack Mode)

במוד זה, הפונקציה ()CollectObservations אינה נחוצה משום שכפי שנאמר בפרק "הפתרון הנבחר", המידע אשר אוסף הרכב מסתפק בשימוש ב-Raycasts, אותן קרני לייזר המספקות מידע לגבי מרחק האובייקטים הסובבים את הרכב (במקרה זה מכשולים/ גבולות המסלול), ואילו פונקציה זו נועדה להוספת פרמטרים נוספים.

שנית, מיפוי output ה-neural network לפעולות קונקרטיות.

בשימוש עם ספריית ML-Agents, מיפוי output ה-neural network נעשה ע"י מימוש הפונקציה OnActionReceived() של המחלקה הבסיסית Agent. בפעולה זו, בנוסף ללקיחת הפעולות, ניתנים ל-Agent (במקרה זה הרכב) בין היתר (הדבר יכול להיעשות בפעולות נוספות) משובים לגבי השינוי בסביבה שיצרה פעולתו- משוב חיובי במידה והפעולה גרמה לתוצאה חיובית (לדוגמא, חנייה מוצלחת) ואילו משוב שלילי במידה והפעולה גרמה לתוצאה שלילית (לדוגמא, התנגשות ברכב אחר).

מוד החנייה (Parking Mode)

כפי שנאמר בפרק "הפתרון הנבחר", הרכב יכול לקחת את הפעולות הבאות:

- תזוזה ימינה/שמאלה/קדימה/אחורה
- היגוי ימינה/ שמאלה

כמו כן, לרכב ניתן משוב חיובי של 1 (המשובים נעים בטווח של בין 1- ל-1) כאשר מבצע חנייה מוצלחת, משוב שלילי של 1- כאשר מתנגש במכונית אחרת או כאשר נופל מפלטפורמת החניון, ומשוב של 1/maxStep- (מינוס אחד חלקי מספר הצעדים בהם יכול לפעול הרכב לפני שהסביבה תתאפס לקראת סבב אימון חדש) כך שלאחר שיגמר פרק האימון, במידה ולא הצליח לחנות (פרק האימון מתאפס ברגע שהרכב חונה בהצלחה)- יקבל משוב שלילי מצטבר של 1-.

```
/// <summary>
/// Specifies the agent behavior at every step based on the provided action.
/// In this case, whether or not the agent shall turn and in which direction,
/// and whether or not the agent should drive and in which direction.
/// </summary>
/// <param name="vectorAction">The action array.</param>
3 references
public override void OnActionReceived(float[] vectorAction)
{
    // Drive according to the neural network decision
    Drive(vectorAction[0]);
    Steer(vectorAction[1]);

    // The agent fell off the platform,
    // ending the episode and assigning a negative reward for falling
    if (transform.position.y < -1f)
    {
        AddReward(-1.0f);
        EndEpisode();
    }

    // Add a -1/maxStep reward to the agent for not parking in order to speed up the parking process.
    // If the agent finished the episode without parking, it will get a maximum total negative reward of -(maxStep/maxStep) = -1
    AddReward(-1 / maxStep);
}
```

```
/// <summary>
/// Drive in the given direction.
/// </summary>
/// <param name="direction">The direction to drive in.</param>
1 reference
public void Drive(float direction)
{
    // Stay in place
    float d = 0f;

    // Drive forward
    if (direction == 1f)
        d = 1f;

    // Go in reverse
    else if (direction == 2f)
        d = -1f;

    // Move the agent in the given direction
    rb.MovePosition(transform.position + transform.forward * d * drivingSpeed * Time.fixedDeltaTime);
}

/// <summary>
/// Rotate in the given direction.
/// </summary>
/// <param name="direction">The rotation direction.</param>
1 reference
public void Steer(float direction)
{
    // Do not rotate
    float d = 0f;

    // Steer left
    if (direction == 1f)
        d = 1f;

    // Steer right
    else if (direction == 2f)
        d = -1f;

    // Rotate in the given direction
    transform.Rotate(transform.up * d * turnAmount * Time.fixedDeltaTime);
}
```

```
/// <summary>
/// Handles the agent collisions; when the agent crashes or parks.
/// </summary>
/// <param name="collision">The collision info.</param>
0 references
void OnCollisionEnter(Collision collision)
{
    // The agent crashed- add a negative reward of 1f for crashing and end the episode
    if (collision.collider.tag == "Obstacle")
    {
        AddReward(-1.0f);
        EndEpisode();
    }

    // The agent parked
    if (collision.collider.tag == "Parking")
    {
        // Increment the agent score
        Score++;
        UIManager.UpdateScore(this);

        // Add a positive reward of 1f for parking and end the episode
        AddReward(1f);
        EndEpisode();
    }
}
```

## מוד מסלול המכשולים (Racetrack Mode)

כפי שנאמר בפרק "הפתרון הנבחר", לרכב ניתנת השליטה בהיגוי (ימינה/שמאלה), כאשר הוא נע קדימה באופן אוטומטי.

כמו כן, לרכב ניתן משוב חיובי של 1/maxStep+ (אחד חלקי מספר הצעדים בהם יכול לפעול הרכב לפני שהסביבה תתאפס לקראת סבב אימון חדש) בכל "צעד", כך שמטרתו הסופית הינה להימנע מהתנגשות עם מכשולים לזמן הרב ביותר (במידה ומצליח לעשות זאת במהלך סבב אימון שלם, יינתן לו משוב חיובי של 1= maxStep/maxStep). התנגשות עם מכשול/ גבולות המסלול תגרור אחריה משוב שלילי של 1- והתחלתו מחדש של פרק האימון הנוכחי.

```
/// <summary>
/// Specifies the agent behavior at every step based on the provided action.
/// In this case, whether or not the agent shall turn and in which direction.
/// </summary>
/// <param name="vectorAction">The action array.</param>
3 references
public override void OnActionReceived(float[] vectorAction)
{
    if (!stopped)
    {
        // The agent moves forward automatically
        MoveForward();

        // Turn based on the neural network output
        Turn(vectorAction[0]);
    }

    // Add a 1/maxStep reward to the agent for not crashing.
    // If the agent finished the episode without crashing, it will get a maximum total reward of maxStep/maxStep = 1
    AddReward(1 / maxStep);
}
```

```
/// <summary>
/// Move the agent forward at a speed of <see cref="forwardSpeed"/>
/// </summary>
1 reference
public void MoveForward()
{
    rbComponent.velocity = transform.forward * forwardSpeed;
}

/// <summary>
/// Turn <see cref="turnAmount"/> degrees in the given direction.
/// </summary>
/// <param name="direction">The direction of the rotation.</param>
1 reference
public void Turn(float direction)
{
    if (direction == 1)
    {
        // Turn right
        transform.Rotate(0, turnAmount, 0);
    }

    else if (direction == 2)
    {
        // Turn left
        transform.Rotate(0, -turnAmount, 0);
    }
}
```

```
/// <summary>
/// Handles the agent collisions; when the agent crashes.
/// </summary>
/// <param name="collision">The collision info.</param>
0 references
void OnCollisionEnter(Collision collision)
{
    if (collision.collider.CompareTag("Obstacle"))
    {
        // Reset the lap counter
        LapsMade = 0;
        UIManager.UpdateLapCount(this);

        // Add a negative reward of 1f to the agent for crashing and end the episode
        AddReward(-1.0f);
        EndEpisode();
    }
}
```

**תיאור המודולים של מערכת התכנה**

במסגרת פיתוח המערכת, נדרשתי להשתמש במודולים שונים כאשר המרכזיים מבניהם הינם-

- Manager.cs/RacetrackManager.cs, אחראים על ניהולו השוטף של המשחק.
- ParkingAgent.cs/CarAgent.cs, מחלקת ה-Agent הרלוונטית לכל מוד. אחראית על איסוף המידע ועל לקיחת פעולותיו השונות של הרכב.
- ParkingUIManager.cs/RacetrackUIManager.cs, דואגות לניהול ה- UI במודים השונים.
- AudioManager.cs, אחראי על ניהול השמע במערכת.

כמו כן, השתמשתי בשתי ספריות עיקריות כאשר המרכזית מבניהן היא Unity ML-Agents, אשר מאפשרת הכנסת אלמנטים של למידת מכונה אל תוך Unity, ובנוסף הספרייה TextMeshPro אשר מקלה על העבודה עם טקסט בUnity.

**תיעוד הקוד**

<u>פירוט מבני הנתונים</u>

| שם | תפקיד |
|---|---|
| CarAgent[] cars<br>ParkingAgent[] cars | קיים בשתי מחלקות הניהול של כל אחד מן המודים. מחזיק הפניה לשתי המכוניות (המשתמש והמחשב) הקיימות במשחק. |
| List<ParkingSpot> | מכיל את כלל מקומות החנייה בחניון |
| Sound[] sounds | מכיל את כל קבצי השמע במערכת. |

עתה, אפרט על כל פונקציה הנמצאת בקוד. בטבלאות הבאות יצוין שם הפונקציה, מה היא מקבלת, מה היא מחזירה ותפקידה.

<u>AudioManager.cs</u>

| כותרת הפונקציה | טענת כניסה | טענת יציאה | תפקיד |
|---|---|---|---|
| void Awake() | - | - | אתחול מערך קבצי האודיו ומימוש תבנית-Singleton |
| void Start() | - | - | נקראת לפני הפריים הראשון, ומתחילה לנגן את מוזיקת המערכת הראשית |
| void Play(string name) | מקבלת את שם קובץ השמע המיועד להפעלה | - | הפעלת קובץ שמע |
| void SetVolume(string name, float volume) | מקבלת את שם הקובץ ועוצמת הקול המיועדת | - | שינוי עוצמת הקול של קובץ שמע |

CarAgent.cs

| כותרת הפונקציה | טענת כניסה | טענת יציאה | תפקיד |
|---|---|---|---|
| void OnEpisodeBegin() | - | - | קביעת התנהגות הרכב בתחילת פרק אימונים חדש |
| void OnActionReceived(float[] vectorAction) | מקבלת את ה- output neural network | - | מיפוי ה- output neural network לביצוע פעולות הרכב, ונתינת משוב בהתאם לביצועי הרכב |
| float[] Heuristic() | - | החזרת מערך הפעולות | דימוי לחיצת מקלדת המשתמש ל-output ה- neural network |
| void MoveForward() | - | - | קידום הרכב באופן אוטומטי |
| void Turn(float direction) | מקבלת את כוון הפנייה | - | סיבוב הרכב |
| void Stop() | - | - | עצירת הרכב |
| void Spawn() | - | - | החזרת הרכב אל נקודת המוצא |
| void OnTriggerEnter(Collider other) | מקבלת את העצם איתו התנגש הרכב | - | טיפול בהתנגשויות שאינן פיזיות (למשל מעבר בנקודת הסיום) |
| void OnCollisionEnter(Collision collision) | מקבלת את פרטי ההתנגשות | - | טיפול בהתנגשויות פיזיות (לדוגמא התנגשות בקיר) |

RaceManager.cs

| כותרת הפונקציה | טענת כניסה | טענת יציאה | תפקיד |
|---|---|---|---|
| void OnStartButtonClick() | - | - | טיפול בלחיצה על כפתור התחלת המשחק |
| IEnumerator StartGame() | - | מחכה לסיום הספירה לאחור לפני התחלת המשחק | התחלת המשחק |
| void EndGame() | - | - | מסיים את המשחק |

RaceTimer.cs

| כותרת הפונקציה | טענת כניסה | טענת יציאה | תפקיד |
|---|---|---|---|
| String ConvertToTimerFormat() | מקבלת את הזמן בשניות | החזרת הזמן בפורמט mm:ss:ms | המרת פורמט הזמן לצורך הצגתו במשחק |
| string GetTotalTime() | - | החזרת זמן השעון הנוכחי בפורמט של mm:ss:ms | החזרת זמן המרוץ הנוכחי בפורמט הרצוי |
| void Update() | - | - | נקראת בכל פריים ומטרתה עדכון שעון המרוץ |

RacetrackUIManager.cs

| כותרת הפונקציה | טענת כניסה | טענת יציאה | תפקיד |
|---|---|---|---|
| void Start() | - | - | נקראת לפני הפריים הראשון למטרות אתחול |
| void UpdateLapCount(CarAgent carAgent) | מקבלת את הרכב לעדכון | - | עדכון מספר ההקפות שביצע הרכב |
| IEnumerator DisplayGameoverScreen(CarAgent user, CarAgent AI) | מקבלת את שני הרכבים המשתתפים במשחק | מחכה מספר שניות לפני הצגת חלק מהסטטיסטיקות | הצגת מסך סיום המשחק |
| void Update() | - | - | נקראת בכל פריים ומגיבה ללחיצת מקש הגורם ESC לעצירת המשחק |
| void Resume() | - | - | המשכת המשחק |
| void Pause() | - | - | עצירת המשחק |
| void OnQuitButtonClick() | - | - | יציאה מהמערכת בלחיצה על כפתור QUIT |
| void MuteAudio() | - | - | השתקת המוזיקה במערכת |

ParkingAgent.cs

| כותרת הפונקציה | טענת כניסה | טענת יציאה | תפקיד |
|---|---|---|---|
| void CollectObservations(VectorSensor sensor) | מקבלת אובייקט המשמש לצורך איסוף מידע והכנסתו כ-input אל תוך ה-neural network | - | איסוף הנתונים המשמשים בסופו של דבר לצורך קבלת ההחלטות |
| void OnEpisodeBegin() | - | - | קביעת התנהגות הרכב בתחילת פרק אימונים חדש |
| void OnActionReceived(float[] vectorAction) | מקבלת את output ה-neural network | - | מיפוי output ה-neural network לביצוע פעולות הרכב, ונתינת משוב בהתאם לביצועי הרכב |
| float[] Heuristic() | - | החזרת מערך הפעולות | דימוי לחיצת מקלדת המשתמש ל-output ה-neural network |
| void Drive(float direction) | מקבלת את כוון הנסיעה | - | הזזת הרכב |
| void Steer(float direction) | מקבלת את כוון הפנייה | - | סיבוב הרכב |
| void OnCollisionEnter(Collision collision) | מקבלת את פרטי ההתנגשות | - | טיפול בהתנגשויות פיזיות (לדוגמא התנגשות ברכב) |

ParkingSpot.cs

| כותרת הפונקציה | טענת כניסה | טענת יציאה | תפקיד |
|---|---|---|---|
| void ActivateCarModel(bool activationStatus) | מקבלת את סטטוס האקטיבציה | - | קביעת נראות הרכב של מקום החנייה |
| void ActivateParkingColliders(bool activationStatus) | מקבלת את סטטוס האקטיבציה | - | קביעת הפעלתם של בודקי ההתנגשות של מקום החנייה |
| void ResetParkingSpot() | - | - | איפוס מקום החנייה- הרכב נראה ובודקי ההתנגשות כבויים |
| void ColorParkingSpot(Material parkingSpotColor) | מקבלת את צבע החנייה | - | צביעת מקום החנייה בצבע מסויים |
| void SetAsChosenParkingSpot() | - | - | התאמת מקום החנייה לכך שנבחר- הרכב מוסתר, ובודקי ההתנגשות מופעלים |
| void OnTriggerStay(Collider other) | מקבלת את הגוף המתנגש | - | טיפול בכניסת הרכב למקום החנייה |

ParkingLot.cs

| כותרת הפונקציה | טענת כניסה | טענת יציאה | תפקיד |
|---|---|---|---|
| void ResetParkingLot(float level) | מקבלת את שלב האימון הנוכחי | - | איפוס החניון ע"פ שלב האימון הנוכחי (דרגת הקושי עולה עם התקדמות הרכב) |
| void SurroundWithCars(ParkingSpot parkingSpot) | מקבלת את מקום החנייה | - | הקפת מקום החנייה במכוניות (כלומר הפיכת מקומות החנייה הצמודים אליו לתפוסים) |
| void SpawnAgent() | - | - | הכנסת הרכב לחניון בזווית ומיקום רנדומליים בתחילת ניסיון חניה חדש |
| Vector3 ChooseRandomPosition() | - | החזרת המקום הנבחר | בחירת מקום רנדומלי בחניון |
| Quaternion ChooseRandomRotation() | - | החזרת הסיבוב הנבחר | בחירת מספר מעלות רנדומלי |
| ParkingSpot GetChosenParkingSpot() | - | החזרת מקום החנייה הנבחר | החזרת מקום החנייה הרנדומלי הנבחר |
| void ResetAllParkingSpots() | - | - | איפוס החניון |
| FindParkingSpotIndex(ParkingSpot parkingSpot) | מקבלת את מקום החנייה | החזרת האינדקס | מיפוי מקום החנייה לאינדקס ברשימה |

ParkingUIManager.cs

| כותרת הפונקציה | טענת כניסה | טענת יציאה | תפקיד |
|---|---|---|---|
| void Start() | - | - | נקראת לפני הפריים הראשון למטרות אתחול |
| void UpdateScore(ParkingAgent parkingAgent) | מקבלת את הרכב לעדכון | - | עדכון מספר תוצאת הרכב |
| IEnumerator DisplayGameoverScreen(ParkingAgent user, ParkingAgent AI) | מקבלת את שני הרכבים המשתתפים במשחק | מחכה מספר שניות לפני הצגת חלק מהסטטיסטיקות | הצגת מסך סיום המשחק |
| void Update() | - | - | נקראת בכל פריים ומגיבה ללחיצת מקש הגורם ESC לעצירת המשחק |
| void Resume() | - | - | המשכת המשחק |
| void Pause() | - | - | עצירת המשחק |
| void OnQuitButtonClick() | - | - | יציאה מהמערכת בלחיצה על כפתור QUIT |
| void MuteAudio() | - | - | השתקת המוזיקה במערכת |

(Parking) Manager.cs

| כותרת הפונקציה | טענת כניסה | טענת יציאה | תפקיד |
|---|---|---|---|
| void OnStartButtonClick() | - | - | טיפול בלחיצה על כפתור התחלת המשחק |
| IEnumerator StartGame() | - | מחכה לסיום הספירה לאחור לפני התחלת המשחק | התחלת המשחק |
| void EndGame() | - | - | מסיים את המשחק |

ParkingTimer.cs

| כותרת הפונקציה | טענת כניסה | טענת יציאה | תפקיד |
|---|---|---|---|
| void Start() | - | - | נקראת לפני הפריים הראשון למטרת אתחול השעון |
| String ConvertToTimerFormat() | מקבלת את הזמן בשניות | החזרת הזמן בפורמט ss:ms | המרת פורמט הזמן לצורך הצגתו במשחק |
| void Update() | - | - | נקראת בכל פריים ומטרתה עדכון שעון המרוץ |

Timer.cs

| כותרת הפונקציה | טענת כניסה | טענת יציאה | תפקיד |
|---|---|---|---|
| String ConvertToTimerFormat() | מקבלת את הזמן בשניות | החזרת הזמן בפורמט הרצוי | המרת מספר שניות לפורמט הזמן הרצוי לצורך הצגתו במשחק |
| string SpaceCharacters(string time) | מקבלת את טקסט הזמן לרווח | החזרת טקסט הזמן מרווח | רווח טקסט הזמן ע"י הכנסת רווח בין כל תו |
| void Show(bool activationStatus) | מקבלת את סטטוס האקטיבציה | - | מראה/מסתירה את השעון בהתאם לסטטוס האקטיבציה |
| void Pause() | - | - | עצירת השעון |

SceneLoader.cs

| כותרת הפונקציה | טענת כניסה | טענת יציאה | תפקיד |
|---|---|---|---|
| void LoadScene(string sceneName) | מקבלת את שם המסך לטעינה | - | קוראת לפונקצית טעינת המסך הא-סינכרונית |
| IEnumerator LoadAsync(string sceneName) | מקבלת את שם המסך לטעינה | מחכה לסיום טעינת המסך | הצגת מסך Loading המציג את התקדמות טעינת המסך בזמן טעינתו |

**השוואת העבודה עם פתרונות ויישומים קיימים**

כאשר נערכת השוואה בין המערכת אותה פיתחתי לבין תוכנות בעלות מטרה דומה, ניכרים יתרונות וחסרונות שונים.

ראשית, יתרון בולט במערכת הינה הפשטות בשימוש אך עם זאת היכולות הרבות שמציעה המערכת. על מנת שחוויית השימוש במערכת תהיה מזמינה ונעימה, גרפיקת המערכת נבנתה ע"י שימוש ב- low poly shapes, אשר נותנות לה מראה אסתטי ונקי ואף משפרות את ביצועיה. יחד עם זאת, כאשר מתבוננים לעומק על יכולותיה של המערכת, ניתן להבחין בפרטים המבדילים אותה ממערכות אחרות. לדוגמא, כחלק המחקר הראשוני לפני הפיתוח נתקלתי במספר סימולציות ביוטיוב המדגימות יכולות דומות לאלה שהרכבים במערכת שלי מספקים, אך ההבדל העיקרי נבדל במוד החנייה- אצלם, הרכב היה מסוגל לחנות אך ורק במקום חנייה אחד קבוע, אך ידעתי שאם ברצוני להמחיש יכולות מציאותיות, יהיה עליי להפוך את בחירת מקום החנייה לבחירה רנדומלית ולכן במהלך כל נסיון חנייה חדש נבחר מקום חניה אקראי בחניון ומספר מעלות בהן מסובב הרכב לפני התחלת הנסיעה.

חסרון מרכזי במערכת לצערי, היא העובדה שאין בידי את המשאבים העצומים שיש לחברות רכב בתהליכי הפיתוח שלהן, ולכן ההדגמה נאלצה להשאר בגדר משחק מחשב בלבד ולא במודל פיזי של רכב אמיתי.

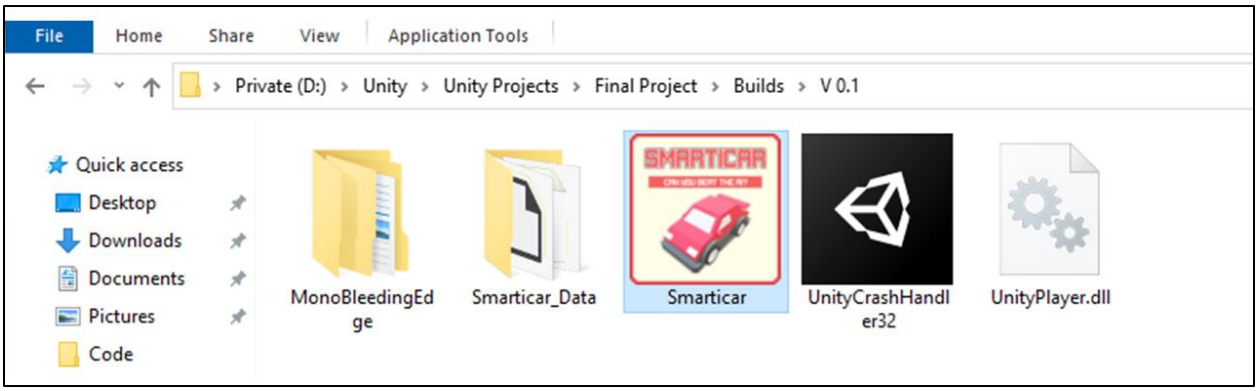
**הערכת הפתרון לעומת התכנון והמלצות לשיפורו**

תחושתי היא שהצלחתי לעמוד בכל המטרות שהצבתי לעצמי בתחילת תהליך הפיתוח ועל כן אני מרוצה מאוד מהתוצר הסופי הנובע מתוך מחקר מעמיק ורציני.

כמו כן, לקראת סוף הפרוייקט כאשר כבר הייתי בשלבי סיום הפיתוח, עלו לי לראש רעיונות נוספים לשיפור והוספת פיצ'רים אך לעומת זאת גיליתי כי הדבר אינו כל כך פשוט בשל מבנה חלק מן המחלקות אותן כתבתי כך שהדבר הכניס אותי לחקור קצת את תחום הנדסת התוכנה ו-design patterns. לאחר תהליך הלימוד והתבוננות בקבצי קוד אשר "כתובים נכון" מבחינה מבנית, למדתי היכן אוכל להשתפר ולתקן את הקוד אך לצערי הדבר היה מאוחר מדי לפרויקט הנוכחי, כך שאיישם אותו בפרוייקט הבא.

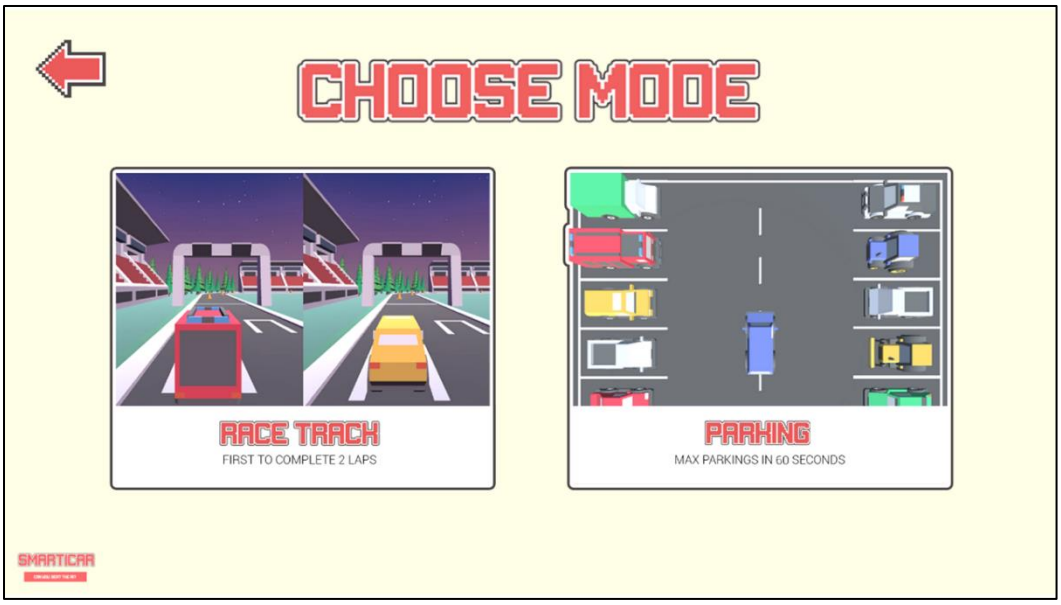
**תיאור של הממשק למשתמש – הוראות הפעלה**

<u>שלב 1)</u> לצורך כניסה למערכת, יש להפעיל את התכנה Smarticar.exe-

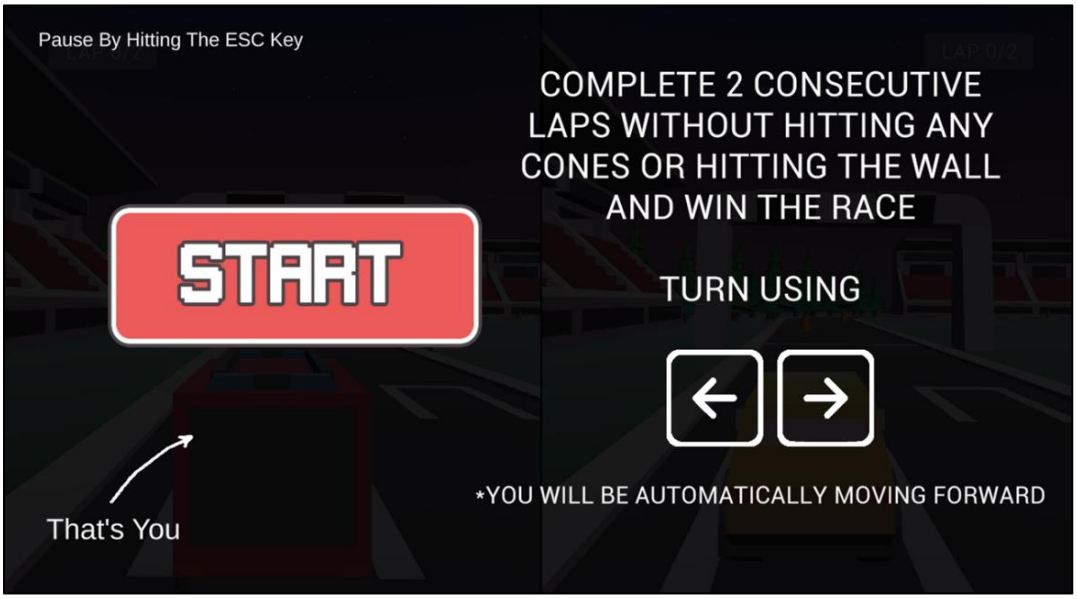



<u>שלב 2)</u> להתחלת המשחק, יש ללחוץ על כפתור START-

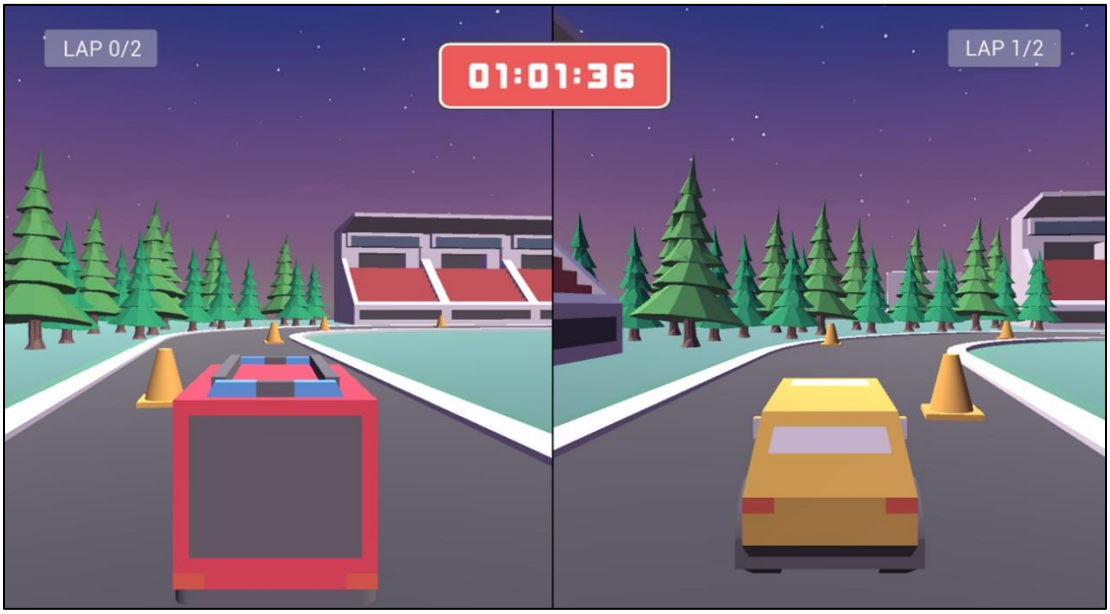
<u>שלב 3)</u> כעת, על המשתמש לבחור את המוד בו הוא רוצה לשחק. לרשותו 2 אופציות- מוד והחנייה ומוד מסלול המכשולים. לצורך ההדגמה נבחר במוד מסלול המכשולים-



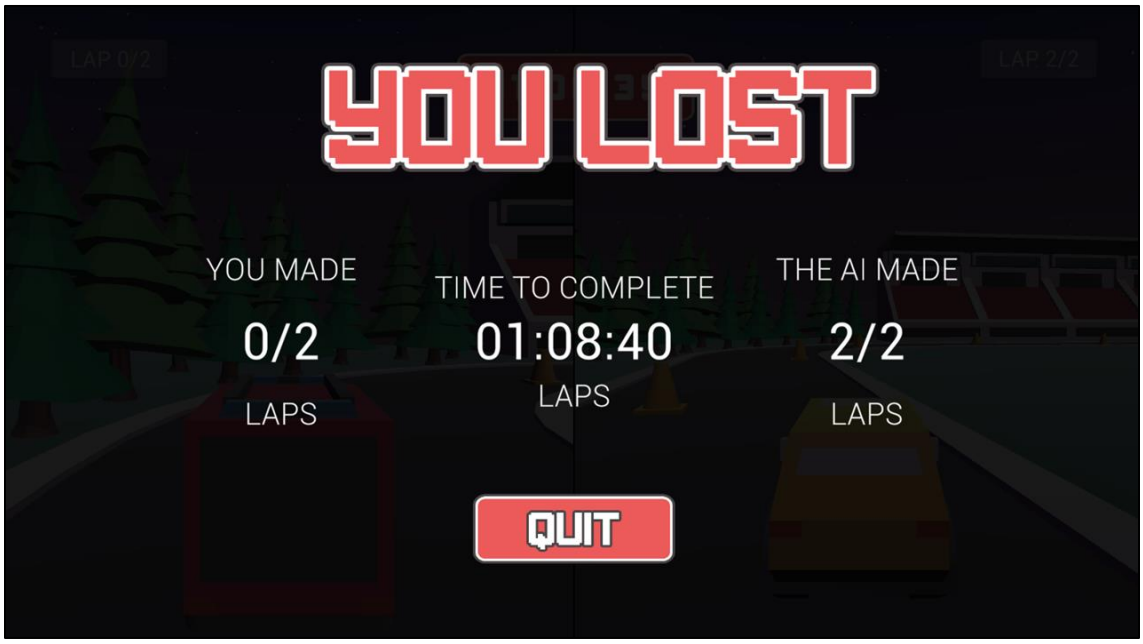

<u>שלב 4)</u> לאחר שהמשתמש בחר במוד הרצוי, יפתח בפניו מסך הוראות המשחק-



שלב 5) לאחר הספירה לאחור המשחק יתחיל וייגמר לאחר שאחד מהנהגים ישלים שני סיבובים רצופים ללא פגיעה ברכב (על מנת לעצור את המשחק על המשתמש ללחוץ על כפתור ESC)-




שלב 6) לאחר סיום המשחק, יוצגו למשתמש סטטיסטיקות המשחק וכפתור יציאה מהמערכת-



**מבט אישי על העבודה ותהליך הפיתוח**

העבודה על הפרוייקט הייתה תהליך מלא באתגרים, כשלונות והצלחות. העבודה דרשה ממני לימוד עצמי בתחומים בהם לא הייתי בקיא כלל, ולכן דרשה ממני לסדר את תהליך הפיתוח בצורה התאפשר לי ניהול נכון של זמני ומשאביי.

למזלי, תהליך המחקר הארוך נערך בזמן תקופת בידוד וירוס הקורונה כך שהיה ברשותי זמן רב להעמיק בנושאים שונים אשר היוו לי בסיס יציב לפיתוח המערכת. הנושאים השונים כללו למידת Unity, למידת הרקע התאורטי בתחום ה-AI וה-Machine Learning והספרייה ML-Agents.

אתגר הפיתוח העיקרי, היה אימון הרכב. הדבר התבטא במציאת הערכים המשמשים לאיסוף מידע הרכב והפעולות אותן יכול לקחת. על מנת להגיע למציאת ערכים אלו, במהלך תהליך המחקר הכנתי פרוייקט דמו (בעזרת הדרכה מהאינטרנט), אשר הדגים לי איזה סוג של מידע נהוג לאסוף בעבודה עם Machine Learning ו-ML-Agents על מנת לקבל את התוצאות האפקטיביות ביותר.

אני מאוד מרוצה מהתוצר הסופי ומהדרך שעברתי על מנת להשיג אותו. למדתי המון, יצא לי להתנסות עם Unity לאחר שנים רבות בהן רציתי לעשות זאת ובעיקר התנסות עם Machine Learning, שאני בטוח שתעזור לי בהמשך. הדבר העיקרי ממנו אני מרוצה, היא העובדה שרכשתי כלים שונים אשר יאפשרו לי עבודה על פרויקטים נוספים בעתיד בצורה יעילה ומסודרת יותר.

## ביבליוגרפיה


- Unity Documentation
  https://docs.unity3d.com/Manual/index.html
- Unity Scripting API
  https://docs.unity3d.com/ScriptReference/
- Unity ML-Agents Toolkit Documentation
  https://github.com/Unity-Technologies/ml-agents/blob/latest_release/docs/Readme.md
- Stack Overflow
  https://stackoverflow.com/
- C# programming guide
  https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/
- Immersive Limit
  https://www.immersivelimit.com/
- Kenny Assets
  https://www.kenney.nl/assets
- Brackeys YouTube Channel
  https://www.youtube.com/user/Brackeys
- 3Blue1Brown YouTube Channel
  https://www.youtube.com/channel/UCYO_jab_esuFRV4b17AJtAw

**קוד התוכנית**

AudioManager.cs

```
using UnityEngine;
using System;

/// <summary>
/// An audio manager for handling audio in the game.
/// </summary>
public class AudioManager : MonoBehaviour
{
    /// <summary>
    /// A <see cref="Sound"/> array consisting of all of the playable sounds.
    /// </summary>
    public Sound[] sounds;

    /// <summary>
    /// Whether or not the audio is currently muted.
    /// </summary>
    public static bool audioIsMuted = false;

    /// <summary>
    /// The audio manager is using the singleton pattern.
    /// This is the audio manager instance.
    /// </summary>
    public static AudioManager instance;

    /// <summary>
    /// Awake is called when the script instance is being loaded.
    /// </summary>
    void Awake()
    {
        // If there is already an instance of the audio manager exists
        if (instance == null)
            instance = this;

        // Destroy the new object
        else
        {
            Destroy(gameObject);
            return;
```

```
    }

    // Keep the audio manager between scenes
    DontDestroyOnLoad(gameObject);

    // Add the appropriate AudioSource components based on the Sound array
    foreach (Sound sound in sounds)
    {
        sound.source = gameObject.AddComponent<AudioSource>();

        sound.source.clip = sound.clip;
        sound.source.volume = sound.volume;
        sound.source.pitch = sound.pitch;
        sound.source.loop = sound.loop;
    }
}

/// <summary>
/// Start is called before the first frame update
/// Play the main theme at the start of the game.
/// </summary>
public void Start()
{
    Play("MainTheme");
}

/// <summary>
/// Play a given audio
/// </summary>
/// <param name="name">The audio to play</param>
public void Play(string name)
{
    Sound s = Array.Find(sounds, sound => sound.name == name);
    if (s == null)
    {
        Debug.LogWarning(string.Format("Sound {0} does not exist", name));
        return;
    }

    s.source.Play();
}

/// <summary>
```

```
    /// Sets the volume of a given audio
    /// </summary>
    /// <param name="name">The audio to set it's volume</param>
    public void SetVolume(string name, float volume)
    {
        Sound s = Array.Find(sounds, sound => sound.name == name);
        if (s == null)
        {
            Debug.LogWarning(string.Format("Sound {0} does not exist", name));
            return;
        }

        s.source.volume = volume;
    }
}
```

CarAgent.cs

```
using UnityEngine;
using MLAgents;

/// <summary>
/// The racetrack agent class.
/// For the ML-Agents Agent documentation-
/// <see href="https://github.com/Unity-Technologies/ml-
agents/blob/master/docs/Learning-Environment-Design-Agents.md"/>
/// </summary>
public class CarAgent : Agent
{
    /// <summary>
    /// True if the agent object corresponds to the user.
    /// </summary>
    public bool isUser;

    // Adds a space of 10 pixels in the inspector
    [Space(10)]

    /// <summary>
    /// The RigidBody component of the agent.
    /// </summary>
    public Rigidbody rbComponent;

    /// <summary>
```

```
    /// The agent's forward speed.
    /// </summary>
    public float forwardSpeed;

    /// <summary>
    /// The degree the agent rotates in a single turn.
    /// </summary>
    public float turnAmount;

    /// <summary>
    /// The reset point for the agent when crashing.
    /// </summary>
    public Vector3 resetPoint;

    [Space(10)]

    /// <summary>
    /// A reference to the race manager.
    /// </summary>
    public RaceManager raceManager;

    /// <summary>
    /// A reference to the UI manager.
    /// </summary>
    public RacetrackUIManager UIManager;

    /// <summary>
    /// The number of laps the agent made in the race.
    /// Will reset when crashing.
    /// </summary>
    [HideInInspector]
    public int LapsMade { get; set; } = 0;

    /// <summary>
    /// Whether or not the agent has passed the checkpoint.
    /// Used to determine if the agent has completed the course when passing through
the starting line.
    /// </summary>
    bool passedCheckpoint;

    /// <summary>
    /// True if the agent stopped.
    /// </summary>
```

```
    bool stopped;

    /// <summary>
    /// Specifies the agent behavior when being reset (due to crashing).
    /// </summary>
    public override void OnEpisodeBegin()
    {
        // Spawn the agent back at the reset point
        Spawn();
    }

    /// <summary>
    /// Specifies the agent behavior at every step based on the provided action.
    /// In this case, whether or not the agent shall turn and in which direction.
    /// </summary>
    /// <param name="vectorAction">The action array.</param>
    public override void OnActionReceived(float[] vectorAction)
    {
        if (!stopped)
        {
            // The agent moves forward automatically
            MoveForward();

            // Turn based on the neural network output
            Turn(vectorAction[0]);
        }

        // Add a 1/maxStep reward to the agent for not crashing.
        // If the agent finished the episode without crashing, it will get a maximum total
reward of maxStep/maxStep = 1
        AddReward(1 / maxStep);
    }

    /// <summary>
    /// When the Agent uses Heuristics, it will call this method every time it needs an
action,
    /// which is based on keyboard input instead of a neural network (lets the user control
the agent).
    /// Since Discrete vector action space is being used, using 1 branch of size 3,
    /// the method shall return a float array of size 1 containing either 0, 1 or 2.
    /// </summary>
    /// <returns>
    /// A float array corresponding to the next action of the Agent
```

```
/// </returns>
public override float[] Heuristic()
{
    // By default, the agent will not turn
    float turnValue = 0;

    if (Input.GetKey(KeyCode.RightArrow))
    {
        turnValue = 1f;
    }

    if (Input.GetKey(KeyCode.LeftArrow))
    {
        turnValue = 2f;
    }

    // Return the action array
    return new float[] { turnValue };
}

/// <summary>
/// Move the agent forward at a speed of <see cref="forwardSpeed"/>
/// </summary>
public void MoveForward()
{
    rbComponent.velocity = transform.forward * forwardSpeed;
}

/// <summary>
/// Turn <see cref="turnAmount"/> degrees in the given direction.
/// </summary>
/// <param name="direction">The direction of the rotation.</param>
public void Turn(float direction)
{
    if (direction == 1)
    {
        // Turn right
        transform.Rotate(0, turnAmount, 0);
    }

    else if (direction == 2)
    {
        // Turn left
```

```
            transform.Rotate(0, -turnAmount, 0);
        }
    }

    /// <summary>
    /// Stop the car by setting <see cref="stopped"/> true.
    /// </summary>
    public void Stop()
    {
        stopped = true;
    }

    /// <summary>
    /// Spawn the agent back at the <see cref="resetPoint"/> facing forward.
    /// </summary>
    public void Spawn()
    {
        transform.localPosition = resetPoint;
        transform.localRotation = Quaternion.Euler(transform.forward);
    }

    /// <summary>
    /// Handles the agent trigger collisions- passing the checkpoint or going through the
starting line.
    /// </summary>
    /// <param name="other">The object the agent collided with.</param>
    void OnTriggerEnter(Collider other)
    {
        // The agent passed the checkpoint
        if (other.CompareTag("LapCheckpoint"))
        {
            passedCheckpoint = true;
        }

        // The agent went through the starting line
        if (other.CompareTag("StartingLine"))
        {
            // If the agent didn't pass the checkpoint,
            // it means that the agent is starting a new lap after a crash
            if (passedCheckpoint)
            {
                // Update the lap counter
                LapsMade++;
```

```
            UIManager.UpdateLapCount(this);

            // If the agent won, the game
            if (LapsMade == 2)
            {
                raceManager.EndGame();
            }
        }

        // Reset passedCheckpoint status
        passedCheckpoint = false;
    }
}

/// <summary>
/// Handles the agent collisions; when the agent crashes.
/// </summary>
/// <param name="collision">The collision info.</param>
void OnCollisionEnter(Collision collision)
{
    if (collision.collider.CompareTag("Obstacle"))
    {
        // Reset the lap counter
        LapsMade = 0;
        UIManager.UpdateLapCount(this);

        // Add a negative reward of 1f to the agent for crashing and end the episode
        AddReward(-1.0f);
        EndEpisode();
    }
}
}
```

Manager.cs

```
using System.Collections;
using UnityEngine;

/// <summary>
/// A class for managing the game.
/// </summary>
public class Manager : MonoBehaviour
{
    /// <summary>
    /// An array holding both agents of the match.
    /// </summary>
    public ParkingAgent[] cars;

    /// <summary>
    /// A reference to the UI manager.
    /// </summary>
    public ParkingUIManager UIManager;

    /// <summary>
    /// Start is called before the first frame update.
    /// </summary>
    void Start()
    {
        // Pause the game for the instructions phase
        Time.timeScale = 0f;
    }

    /// <summary>
    /// Start the game when clicking on the start button.
    /// </summary>
    public void OnStartButtonClick()
    {
        StartCoroutine(StartGame());
    }

    /// <summary>
    /// Starts the game by disabling the instructions panel and starting the countdown.
    /// </summary>
    /// <returns>
    /// Waits for the countdown to finish.
    /// </returns>
```

```
    IEnumerator StartGame()
    {
        // Hide instructions
        UIManager.instructions.SetActive(false);

        // Start countdown by subtracting from the total time and waits for subtraction to
finish
        UIManager.countdownTimer.gameObject.SetActive(true);
        float totalTime = 3f;
        while (totalTime > 0)
        {
            if (totalTime < 1) UIManager.countdownTimer.text = "GO";
            else UIManager.countdownTimer.text = totalTime.ToString("0");
            totalTime -= 0.05f;
            yield return new WaitForSecondsRealtime(0.05f);
        }

        // Hide all panels
        UIManager.countdownTimer.gameObject.SetActive(false);
        UIManager.darkPanel.SetActive(false);

        // Show timer
        UIManager.gameTimer.Show(true);

        // Start the game by resetting timeScale to 1f
        Time.timeScale = 1f;

        // the player entered the gameplay phase
        ParkingUIManager.duringGameplay = true;
    }

    /// <summary>
    /// Ends the game.
    /// </summary>
    public void EndGame()
    {
        // Pause the game
        Time.timeScale = 0f;

        // Display gameover screen
        StartCoroutine(UIManager.DisplayGameoverScreen(cars[0], cars[1]));
    }
}
```

ParkingAgent.cs

```
using UnityEngine;
using MLAgents;
using MLAgents.Sensors;

/// <summary>
/// The parking agent class.
/// For the ML-Agents Agent documentation-
/// <see href="https://github.com/Unity-Technologies/ml-
agents/blob/master/docs/Learning-Environment-Design-Agents.md"/>
/// </summary>
public class ParkingAgent : Agent
{
    /// <summary>
    /// True if the agent object corresponds to the user.
    /// </summary>
    public bool isUser;

    // Adds a space of 10 pixels in the inspector
    [Space(10)]

    /// <summary>
    /// The RigidBody component of the agent.
    /// </summary>
    public Rigidbody rb;

    /// <summary>
    /// The agent's driving speed.
    /// </summary>
    public float drivingSpeed;

    /// <summary>
    /// The degree the agent rotates in a single turn.
    /// </summary>
    public float turnAmount;

    [Space(10)]

    /// <summary>
    /// The parking lot the agent drives in.
    /// </summary>
    public ParkingLot parkingLot;
```

```
    /// <summary>
    /// A reference to the UI manager.
    /// </summary>
    public ParkingUIManager UIManager;

    /// <summary>
    /// The score of the agent.
    /// </summary>
    [HideInInspector]
    public int Score { get; set; } = 0;

    /// <summary>
    /// The selected parking spot
    /// </summary>
    ParkingSpot parkingSpot;

    /// <summary>
    /// Collects the agent vector observations in order to feed the neural network policy.
    /// The agent calculate it's distance from the parking spot, the direction to the parking
spot,
    /// and the direction the car is facing.
    /// </summary>
    /// <param name="sensor">A <see cref="VectorSensor"/> object for adding
observations.</param>
    public override void CollectObservations(VectorSensor sensor)
    {
        Vector3 parkingSpotPosition = parkingSpot.transform.position;


        // The distance from the parking spot
        sensor.AddObservation(Vector3.Distance(transform.position,
parkingSpotPosition));

        // The direction to the parking spot
        sensor.AddObservation((parkingSpotPosition - transform.position).normalized);

        // The direction the car is facing
        sensor.AddObservation(transform.forward);
    }

    /// <summary>
```

```
    /// Specifies the agent behavior when being reset (due to crashing or falling off the
platform).
    /// </summary>
    public override void OnEpisodeBegin()
    {
        // Reset the parking lot based on the current level of the curricula.
        // Used when training using curriculum training

parkingLot.ResetParkingLot(Academy.Instance.FloatProperties.GetPropertyWithDefault
("level", 9f));

        // Store the chosen random parking spot
        parkingSpot = parkingLot.GetChosenParkingSpot();
    }

    /// <summary>
    /// Specifies the agent behavior at every step based on the provided action.
    /// In this case, whether or not the agent shall turn and in which direction,
    /// and whether or not the agent should drive and in which direction.
    /// </summary>
    /// <param name="vectorAction">The action array.</param>
    public override void OnActionReceived(float[] vectorAction)
    {
        // Drive according to the neural network decision
        Drive(vectorAction[0]);
        Steer(vectorAction[1]);

        // The agent fell off the platform,
        // ending the episode and assigning a negative reward for falling
        if (transform.position.y < -1f)
        {
            AddReward(-1.0f);
            EndEpisode();
        }

        // Add a -1/maxStep reward to the agent for not parking in order to speed up the
parking process.
        // If the agent finished the episode without parking, it will get a maximum total
negative reward of -(maxStep/maxStep) = -1
        AddReward(-1 / maxStep);
    }

    /// <summary>
```

```
    /// When the Agent uses Heuristics, it will call this method every time it needs an
action,
    /// which is based on keyboard input instead of a neural network (lets the user control
the agent).
    /// Since Discrete vector action space is being used, using 2 branch of size 3,
    /// the method shall return a float array of size 2 each slot containing either 0, 1 or 2.
    /// </summary>
    /// <returns>
    /// A float array corresponding to the next action of the Agent
    /// </returns>
    public override float[] Heuristic()
    {
        // By default, the agent will stay in place
        float drivingDirection = 0f;

        if (Input.GetKey(KeyCode.UpArrow))
            drivingDirection = 1f;

        else if (Input.GetKey(KeyCode.DownArrow))
            drivingDirection = 2f;

        // By default, the agent will not turn
        float steeringDirection = 0f;

        if (Input.GetKey(KeyCode.RightArrow))
            steeringDirection = 1f;

        else if (Input.GetKey(KeyCode.LeftArrow))
            steeringDirection = 2f;

        // Return the action array
        return new float[] { drivingDirection, steeringDirection };
    }

    /// <summary>
    /// Drive in the given direction.
    /// </summary>
    /// <param name="direction">The direction to drive in.</param>
    public void Drive(float direction)
    {
        // Stay in place
        float d = 0f;
```

```
        // Drive forward
        if (direction == 1f)
            d = 1f;

        // Go in reverse
        else if (direction == 2f)
            d = -1f;

        // Move the agent in the given direction
        rb.MovePosition(transform.position + transform.forward * d * drivingSpeed *
Time.fixedDeltaTime);
    }

    /// <summary>
    /// Rotate in the given direction.
    /// </summary>
    /// <param name="direction">The rotation direction.</param>
    public void Steer(float direction)
    {
        // Do not rotate
        float d = 0f;

        // Steer left
        if (direction == 1f)
            d = 1f;

        // Steer right
        else if (direction == 2f)
            d = -1f;

        // Rotate in the given direction
        transform.Rotate(transform.up * d * turnAmount * Time.fixedDeltaTime);
    }

    /// <summary>
    /// Handles the agent collisions; when the agent crashes or parks.
    /// </summary>
    /// <param name="collision">The collision info.</param>
    void OnCollisionEnter(Collision collision)
    {
        // The agent crashed- add a negative reward of 1f for crashing and end the episode
        if (collision.collider.tag == "Obstacle")
        {
```

```
            AddReward(-1.0f);
            EndEpisode();
        }

        // The agent parked
        if (collision.collider.tag == "Parking")
        {
            // Increment the agent score
            Score++;
            UIManager.UpdateScore(this);

            // Add a positive reward of 1f for parking and end the episode
            AddReward(1f);
            EndEpisode();
        }
    }
}
```

ParkingLot.cs

```
using System.Collections.Generic;
using UnityEngine;
using MLAgents;

/// <summary>
/// Used for handling a parking lot and set it states based on the current level of the
curriculum
/// (the parking agent were trained using curriculum training).
/// </summary>
public class ParkingLot : MonoBehaviour
{
    /// <summary>
    /// A list of all of the parking lot parking spots.
    /// </summary>
    public List<ParkingSpot> parkingSpots;

    /// <summary>
    /// The agent of the parking lot.
    /// </summary>
    public ParkingAgent parkingAgent;

    /// <summary>
    /// The chosen random parking spot.
```

```
    /// </summary>
    ParkingSpot randomParkingSpot;

    /// <summary>
    /// Reset the parking lot based on the current level of the curriculum.
    /// </summary>
    /// <param name="level">The current level of the curriculum.</param>
    public void ResetParkingLot(float level)
    {
        ResetAllParkingSpots();

        // An empty parking lot, chosen parking spots are only from the left side
        if (level == 0f)
        {
            randomParkingSpot = parkingSpots[Random.Range(0, parkingSpots.Count/2)];
            randomParkingSpot.SetAsChosenParkingSpot();

            foreach (ParkingSpot parkingSpot in parkingSpots)
            {
                if (parkingSpot != randomParkingSpot)
                    parkingSpot.ActivateCarModel(false);
            }
        }

        // An empty parking lot, chosen parking spots are only from the right side
        else if (level == 1f)
        {
            randomParkingSpot = parkingSpots[Random.Range(parkingSpots.Count/2,
parkingSpots.Count)];
            randomParkingSpot.SetAsChosenParkingSpot();

            foreach (ParkingSpot parkingSpot in parkingSpots)
            {
                if (parkingSpot != randomParkingSpot)
                    parkingSpot.ActivateCarModel(false);
            }
        }

        // An empty parking lot, chosen parking spots are from both sides
        else if (level == 2f)
        {
            // Select a random parking spot at the parking lot
            randomParkingSpot = parkingSpots[Random.Range(0, parkingSpots.Count)];
```

```
            randomParkingSpot.SetAsChosenParkingSpot();

            foreach (ParkingSpot parkingSpot in parkingSpots)
            {
                if (parkingSpot != randomParkingSpot)
                    parkingSpot.ActivateCarModel(false);
            }
        }

        // Chosen parking spots are only from the left side and are surrounded with cars
        else if (level == 3f)
        {
            randomParkingSpot = parkingSpots[Random.Range(0, parkingSpots.Count / 2)];
            randomParkingSpot.SetAsChosenParkingSpot();

            foreach (ParkingSpot parkingSpot in parkingSpots)
            {
                if (parkingSpot != randomParkingSpot)
                    parkingSpot.ActivateCarModel(false);
            }

            SurroundWithCars(randomParkingSpot);
        }

        // Chosen parking spots are only from the right side and are surrounded with cars
        else if (level == 4f)
        {
            randomParkingSpot = parkingSpots[Random.Range(parkingSpots.Count / 2,
parkingSpots.Count)];
            randomParkingSpot.SetAsChosenParkingSpot();

            foreach (ParkingSpot parkingSpot in parkingSpots)
            {
                if (parkingSpot != randomParkingSpot)
                    parkingSpot.ActivateCarModel(false);
            }

            SurroundWithCars(randomParkingSpot);
        }

        // Chosen parking spots are from both sides and are surrounded with cars
        else if (level == 5f)
        {
```

```
            randomParkingSpot = parkingSpots[Random.Range(0, parkingSpots.Count)];
            randomParkingSpot.SetAsChosenParkingSpot();

            foreach (ParkingSpot parkingSpot in parkingSpots)
            {
                if (parkingSpot != randomParkingSpot)
                    parkingSpot.ActivateCarModel(false);
            }

            SurroundWithCars(randomParkingSpot);
        }

        // Chosen parking spots are only from the left side.
        // they are surrounded with cars + the other side of the parking lot is full of cars
        else if (level == 6f)
        {
            randomParkingSpot = parkingSpots[Random.Range(0, parkingSpots.Count / 2)];
            randomParkingSpot.SetAsChosenParkingSpot();

            foreach (ParkingSpot parkingSpot in parkingSpots)
            {
                if (parkingSpot.side == randomParkingSpot.side && parkingSpot !=
randomParkingSpot)
                    parkingSpot.ActivateCarModel(false);
            }

            SurroundWithCars(randomParkingSpot);
        }

        // Chosen parking spots are only from the right side.
        // they are surrounded with cars + the other side of the parking lot is full of cars
        else if (level == 7f)
        {
            randomParkingSpot = parkingSpots[Random.Range(parkingSpots.Count / 2,
parkingSpots.Count)];
            randomParkingSpot.SetAsChosenParkingSpot();

            foreach (ParkingSpot parkingSpot in parkingSpots)
            {
                if (parkingSpot.side == randomParkingSpot.side && parkingSpot !=
randomParkingSpot)
                    parkingSpot.ActivateCarModel(false);
            }
```

```
            SurroundWithCars(randomParkingSpot);
        }

        // Chosen parking spots are both sides.
        // they are surrounded with cars + the other side of the parking lot is full of cars
        else if (level == 8f)
        {
            randomParkingSpot = parkingSpots[Random.Range(0, parkingSpots.Count)];
            randomParkingSpot.SetAsChosenParkingSpot();

            foreach (ParkingSpot parkingSpot in parkingSpots)
            {
                if (parkingSpot.side == randomParkingSpot.side && parkingSpot !=
randomParkingSpot)
                    parkingSpot.ActivateCarModel(false);
            }

            SurroundWithCars(randomParkingSpot);
        }

        // The default level 9, a full parking lot
        else
        {
            randomParkingSpot = parkingSpots[Random.Range(0, parkingSpots.Count)];
            randomParkingSpot.SetAsChosenParkingSpot();
        }

        // Spawn the agent into the parking lot
        SpawnAgent();
    }

    /// <summary>
    /// Surround the selected parking spot with cars by activating the closest parking
spots car models.
    /// </summary>
    /// <param name="parkingSpot">The parking spot to surround.</param>
    public void SurroundWithCars(ParkingSpot parkingSpot)
    {
        int parkingSpotIndex = FindParkingSpotIndex(parkingSpot);

        // Handling the corner parking spots
        if (parkingSpotIndex == 0)
```

```
            parkingSpots[1].ActivateCarModel(true);

        else if (parkingSpotIndex == 5)
            parkingSpots[4].ActivateCarModel(true);

        else if (parkingSpotIndex == 6)
            parkingSpots[7].ActivateCarModel(true);

        else if (parkingSpotIndex == 11)
            parkingSpots[10].ActivateCarModel(true);

        else
        {
            parkingSpots[parkingSpotIndex + 1].ActivateCarModel(true);
            parkingSpots[parkingSpotIndex - 1].ActivateCarModel(true);
        }
    }

    /// <summary>
    /// Spawn the agent into the parking lot.
    /// </summary>
    public void SpawnAgent()
    {
        parkingAgent.rb.velocity = Vector3.zero;
        parkingAgent.rb.angularVelocity = Vector3.zero;

        parkingAgent.transform.localPosition = ChooseRandomPosition();
        parkingAgent.transform.localRotation = ChooseRandomRotation();
    }

    /// <summary>
    /// Choose a random position in the parking spot to spawn the agent in.
    /// </summary>
    /// <returns>
    /// The chosen position.
    /// </returns>
    public Vector3 ChooseRandomPosition()
    {
        float zSpawnPosition =
Academy.Instance.FloatProperties.GetPropertyWithDefault("spawn_range", 1f);
        float randomXPosition = Random.Range(-2.5f, 2.5f);
        float randomZPosition = Random.Range(-zSpawnPosition, zSpawnPosition);
```

```
    return new Vector3(randomXPosition, 0.5f, randomZPosition);
}

/// <summary>
/// Choose a random rotation to spawn the agent in.
/// </summary>
/// <returns>
/// The chosen rotation.
/// </returns>
public Quaternion ChooseRandomRotation()
{
    return Quaternion.Euler(0f, Random.Range(0f, 360f), 0f);
}

/// <summary>
/// Get the chosen parking spot.
/// </summary>
/// <returns>
/// The chosen parking spot
/// </returns>
public ParkingSpot GetChosenParkingSpot()
{
    return randomParkingSpot;
}

/// <summary>
/// Reset all parking spots into their standart state.
/// </summary>
void ResetAllParkingSpots()
{
    foreach (ParkingSpot parkingSpot in parkingSpots)
        parkingSpot.ResetParkingSpot();
}

/// <summary>
/// Return the index of a given parking spot in the parking spots list.
/// </summary>
/// <returns>
/// The index of the parking spot.
/// </returns>
/// <param name="parkingSpot">The parking spot to find it's index.</param>
int FindParkingSpotIndex(ParkingSpot parkingSpot)
{
```

```
        for (int i =0; i < parkingSpots.Count; i++)
        {
            if (parkingSpots[i] == parkingSpot)
                return i;
        }

        // The item does not exist in the list
        return -1;
    }
}
```

ParkingSpot.cs

```
using UnityEngine;

/// <summary>
/// A class representing a single parking spot
/// </summary>
public class ParkingSpot : MonoBehaviour
{
    /// <summary>
    /// The collider of the parking spot used to check if the agent parked.
    /// </summary>
    public BoxCollider parkingCollider;

    /// <summary>
    /// The trigger collider of the parking spot used to determine
    /// if the user entered the parking spot in order to color it in green.
    /// </summary>
    public BoxCollider parkingTrigger;

    /// <summary>
    /// The car model of the parking spot
    /// </summary>
    public GameObject carModel;

    /// <summary>
    /// The lines of the parking spot
    /// </summary>
    public MeshRenderer[] parkingSpotLines;

    /// <summary>
    /// The side of the parking spot in the parking lot
```

```
/// </summary>
public string side;

/// <summary>
/// Whether or not the parking spot was chosen as the random parking spot
/// </summary>
public bool isChosen;

/// <summary>
/// The parking lines colors-
/// standart = white, marked = red and successful = green.
/// </summary>
public Material standartParkingSpot;
public Material markedParkingSpot;
public Material successfulParkingSpot;

/// <summary>
/// Set the activation status of the parking spot car model.
/// </summary>
/// <param name="activationStatus">true=activate, false=deactivate.</param>
public void ActivateCarModel(bool activationStatus)
{
    carModel.SetActive(activationStatus);
}

/// <summary>
/// Set the activation status of the parking spot colliders.
/// </summary>
/// <param name="activationStatus">true=activate, false=deactivate.</param>
public void ActivateParkingColliders(bool activationStatus)
{
    parkingCollider.enabled = activationStatus;
    parkingTrigger.enabled = activationStatus;
}

/// <summary>
/// Reset the parking spot to it's default state-
/// car model is activated, colliders are deactivated and the lines are colored white.
/// </summary>
public void ResetParkingSpot()
{
    ActivateCarModel(true);
    ActivateParkingColliders(false);
```

```
        isChosen = false;
        parkingSpotLines[0].enabled = false;
        ColorParkingSpot(standartParkingSpot);
    }

    /// <summary>
    /// Color the parking spot lines in a given color
    /// </summary>
    /// <param name="parkingSpotColor">the color to change to</param>
    public void ColorParkingSpot(Material parkingSpotColor)
    {
        foreach(MeshRenderer parkingSpotLine in parkingSpotLines)
            parkingSpotLine.material = parkingSpotColor;
    }

    /// <summary>
    /// Set the parking spot as chosen-
    /// car model is deactivated, colliders are activated and the lines are colored red.
    /// </summary>
    public void SetAsChosenParkingSpot()
    {
        isChosen = true;
        ActivateCarModel(false);
        ActivateParkingColliders(true);
        parkingSpotLines[0].enabled = true;
        ColorParkingSpot(markedParkingSpot);
    }

    /// <summary>
    /// When the agent enters the chosen parking spot, color it green.
    /// </summary>
    /// <param name="other">The agent collider</param>
    public void OnTriggerStay(Collider other)
    {
        if (isChosen)
            ColorParkingSpot(successfulParkingSpot);
    }
}
```

ParkingTimer

```
using UnityEngine;
using System;

/// <summary>
/// A class representing a timer designed specific for the Parking mode.
/// The parking timer is acting as a countdown timer starts at 60 seconds.
/// </summary>
public class ParkingTimer : Timer
{
    /// <summary>
    /// The game manager of the game
    /// </summary>
    public Manager gameManager;

    /// <summary>
    /// Start is called before the first frame update
    /// </summary>
    void Start()
    {
        // Reset the clock at 60 seconds
        totalTime = 60f;
    }

    /// <summary>
    /// Convert a float type time into a format of ss:ms.
    /// </summary>
    /// <returns>
    /// The time in the desired format.
    /// </returns>
    /// <param name="timePassed">The time to convert.</param>
    public override string ConvertToTimerFormat(float timePassed)
    {
        return
SpaceCharacters(TimeSpan.FromMinutes(timePassed).ToString().Substring(3, 5));
    }

    /// <summary>
    /// Update is called once per frame.
    /// </summary>
    void Update()
    {
```

```
        // Update the timer text with the current total time
        timerText.text = ConvertToTimerFormat(totalTime);
        if (totalTime - Time.deltaTime >= 0) totalTime -= Time.deltaTime;

        // Making sure the timer doesn't go below 0
        else
        {
            totalTime = 0;
            Time.timeScale = 0f;
            gameManager.EndGame();
        }
    }
}
```

ParkingUIManager.cs

```
using System.Collections;
using UnityEngine;
using TMPro;

public class ParkingUIManager : MonoBehaviour
{
    /// <summary>
    /// The dark panel much of the UI is displayed on (e.i instructions, gameover).
    /// </summary>
    public GameObject darkPanel;

    // Adds a header in the inspector
    [Header("Starting")]

    /// <summary>
    /// The instructions game object consisting of the tutorial text and the images.
    /// </summary>
    public GameObject instructions;

    /// <summary>
    /// The starting countdown timer text.
    /// </summary>
    public TextMeshProUGUI countdownTimer;

    [Header("Gameplay")]

    /// <summary>
```

```
    /// The timer of the game.
    /// </summary>
    public ParkingTimer gameTimer;

    /// <summary>
    /// The user score (number of parkings).
    /// </summary>
    public TextMeshPro userScore;

    /// <summary>
    /// /// The AI score (number of parkings).
    /// </summary>
    public TextMeshPro AIScore;

    /// <summary>
    /// A pause menu appearing when pressing ESC.
    /// </summary>
    public GameObject pauseMenu;

    /// <summary>
    /// The mute button text.
    /// </summary>
    public TextMeshProUGUI muteButtonText;

    /// <summary>
    /// Whether or not the user is currently playing (not in the instructions phase or the
game over phase).
    /// </summary>
    [HideInInspector]
    public static bool duringGameplay;

    /// <summary>
    /// Set to true if the game is currently paused.
    /// </summary>
    [HideInInspector]
    public static bool gameIsPaused = false;

    [Header("Game Over")]

    /// <summary>
    /// The game over screen default text.
    /// </summary>
    public GameObject gameoverDefaultText;
```

```
/// <summary>
/// The "YOU WON" image displayed when the user wins.
/// </summary>
public GameObject youWonImage;

/// <summary>
/// The "YOU LOST" image displayed when the user wins.
/// </summary>
public GameObject youLostImage;

/// <summary>
/// the game over text of the user score
/// </summary>
public TextMeshProUGUI userTotalParkings;

/// <summary>
/// the game over text of the AI score
/// </summary>
public TextMeshProUGUI AITotalParkings;

/// <summary>
/// The game over quit button.
/// </summary>
public GameObject quitButton;

/// <summary>
/// A reference to the audio manager.
/// </summary>
[Space(10)]
public AudioManager audioManager;

/// <summary>
/// Start is called before the first frame update
/// </summary>
void Start()
{
    audioManager = FindObjectOfType<AudioManager>();
}

/// <summary>
/// Update the score of a given agent.
/// </summary>
```

```
/// <param name="parkingAgent">The agent to update it's score count.</param>
public void UpdateScore(ParkingAgent parkingAgent)
{
    if (parkingAgent.isUser)
    {
        userScore.text = parkingAgent.Score.ToString();
    }

    else
    {
        AIScore.text = parkingAgent.Score.ToString();
    }
}

/// <summary>
/// Display the game over screen.
/// </summary>
/// <returns>
/// Waits before displaying stats.
/// </returns>
/// <param name="user">The user agent.</param>
/// <param name="AI">The AI agent.</param>
public IEnumerator DisplayGameoverScreen(ParkingAgent user, ParkingAgent AI)
{
    duringGameplay = false;

    // Wait 1 second before displaying the dark screen and the game over text
    yield return new WaitForSecondsRealtime(1);
    gameTimer.Show(false);
    darkPanel.SetActive(true);
    gameoverDefaultText.SetActive(true);

    // Display the correct image according to the user winning status
    if (user.Score > AI.Score) youWonImage.SetActive(true);
    else youLostImage.SetActive(true);

    // Wait 1 second before displaying the game stat
    yield return new WaitForSecondsRealtime(1);
    userTotalParkings.text = userScore.text;
    userTotalParkings.gameObject.SetActive(true);

    yield return new WaitForSecondsRealtime(1);
    AITotalParkings.text = AIScore.text;
```

```
        AITotalParkings.gameObject.SetActive(true);

        // Wait 1 second and display the quit button
        yield return new WaitForSecondsRealtime(1);
        quitButton.SetActive(true);
    }

    /// <summary>
    /// Update is called once per frame.
    /// </summary>
    void Update()
    {
        // Pause only if the player is during the game (not in the instructions or gameover
phase)
        if (Input.GetKeyDown(KeyCode.Escape) && duringGameplay)
        {
            if (gameIsPaused) Resume();

            else
            {
                Pause();
            }
        }
    }

    public void Resume()
    {
        pauseMenu.SetActive(false);
        darkPanel.SetActive(false);
        Time.timeScale = 1f;
        gameIsPaused = false;
    }

    public void Pause()
    {
        darkPanel.SetActive(true);
        pauseMenu.SetActive(true);
        Time.timeScale = 0;
        gameIsPaused = true;
    }

    /// <summary>
    /// Quit the application when pressing on the quit button.
```

```
    /// </summary>
    public void OnQuitButtonClick()
    {
        Application.Quit();
    }

    public void MuteAudio()
    {
        if (AudioManager.audioIsMuted)
        {
            audioManager.SetVolume("MainTheme", 0.5f);
            AudioManager.audioIsMuted = false;
            muteButtonText.text = "MUTE";
        }

        else
        {
            audioManager.SetVolume("MainTheme", 0);
            AudioManager.audioIsMuted = true;
            muteButtonText.text = "UNMUTE";
        }
    }
}
```

RaceManager.cs

```
using System.Collections;
using UnityEngine;

/// <summary>
/// A class for managing the race.
/// </summary>
public class RaceManager : MonoBehaviour
{
    /// <summary>
    /// An array holding a reference for the user and the AI agents.
    /// </summary>
    public CarAgent[] cars;

    /// <summary>
    /// A reference to the UI manager.
    /// </summary>
```

```
    public RacetrackUIManager UIManager;

    /// <summary>
    /// The race timer.
    /// </summary>
    public RaceTimer raceTimer;

    /// <summary>
    /// Start is called before the first frame update
    /// </summary>
    void Start()
    {
        // Pause the game for the instructions phase
        Time.timeScale = 0f;
    }

    /// <summary>
    /// Starts the game when the user clicks on the start button.
    /// </summary>
    public void OnStartButtonClick()
    {
        StartCoroutine(StartGame());
    }

    /// <summary>
    /// Starts the game by disabling the instructions panel and starting the countdown.
    /// </summary>
    /// <returns>
    /// Waits for the countdown to finish.
    /// </returns>
    IEnumerator StartGame()
    {
        // Hide instructions
        UIManager.instructions.SetActive(false);

        // Start countdown by subtracting from the total time and waits for subtraction to
finish
        UIManager.countdownTimer.gameObject.SetActive(true);
        float timeLeftForCountdown = 3f;

        while (timeLeftForCountdown > 0)
        {
            if (timeLeftForCountdown < 1) UIManager.countdownTimer.text = "GO";
```

```
            else UIManager.countdownTimer.text = timeLeftForCountdown.ToString("0");

            timeLeftForCountdown -= 0.05f;
            yield return new WaitForSecondsRealtime(0.05f);
        }

        // Hide all panels
        UIManager.countdownTimer.gameObject.SetActive(false);
        UIManager.darkPanel.SetActive(false);

        // Show timer
        raceTimer.Show(true);

        // Start the game by resetting timeScale to 1f
        Time.timeScale = 1f;

        // the player entered the gameplay phase
        RacetrackUIManager.duringGameplay = true;
    }

    /// <summary>
    /// Ends the game.
    /// </summary>
    public void EndGame()
    {
        // Stop agents and timer
        foreach (CarAgent car in cars) car.Stop();
        raceTimer.Pause();

        // Display gameover screen
        StartCoroutine(UIManager.DisplayGameoverScreen(cars[0], cars[1]));
    }
}
```

RaceTimer.cs

```
using UnityEngine;
using System;

/// <summary>
/// A class representing a timer designed specific for the Racetrack mode.
/// The racetrack timer is acting as a stopwatch stopping when the game ends.
/// </summary>
public class RaceTimer : Timer
{
    /// <summary>
    /// Start is called before the first frame update
    /// </summary>
    void Start()
    {
        totalTime = 0f;
    }

    /// <summary>
    /// Converts a float type time into a format of mm:ss:ms.
    /// </summary>
    /// <returns>
    /// The time in the desired format.
    /// </returns>
    /// <param name="time">The time to convert.</param>
    public override string ConvertToTimerFormat(float time)
    {
        return TimeSpan.FromMinutes(time).ToString().Substring(0, 8);
    }

    /// <summary>
    /// Returns the total time of the timer in the format of mm:ss:ms.
    /// </summary>
    /// <returns>
    /// The time in the desired format.
    /// </returns>
    public string GetTotalTime()
    {
        return ConvertToTimerFormat(totalTime);
    }

    /// <summary>
```

```
    /// Update is called once per frame
    /// </summary>
    void Update()
    {
        if (!paused)
        {
            // Update the timer text with the current total time
            timerText.text = SpaceCharacters(ConvertToTimerFormat(totalTime));
            totalTime += Time.deltaTime;
        }
    }
}
```

RacetrackUIManager.cs

```
using System.Collections;
using UnityEngine;
using TMPro;

/// <summary>
/// A Manager for handling all of the UI stuff in the Racetrack mode.
/// </summary>
public class RacetrackUIManager : MonoBehaviour
{
    /// <summary>
    /// The dark panel much of the UI is displayed on (e.i instructions, gameover).
    /// </summary>
    public GameObject darkPanel;

    // Adds a header in the inspector
    [Header("Starting")]

    /// <summary>
    /// The instructions game object consisting of the tutorial text and the images.
    /// </summary>
    public GameObject instructions;

    /// <summary>
    /// The starting countdown timer text.
    /// </summary>
    public TextMeshProUGUI countdownTimer;
```

```
    [Header("Gameplay")]

    /// <summary>
    /// The timer of the race.
    /// </summary>
    public RaceTimer raceTimer;

    /// <summary>
    /// The user lap count text
    /// </summary>
    public TextMeshProUGUI userLapCount;

    /// <summary>
    /// The AI lap count text
    /// </summary>
    public TextMeshProUGUI AILapCount;

    /// <summary>
    /// A pause menu appearing when pressing ESC.
    /// </summary>
    public GameObject pauseMenu;

    /// <summary>
    /// The mute button text.
    /// </summary>
    public TextMeshProUGUI muteButtonText;

    /// <summary>
    /// Whether or not the user is currently playing (not in the instructions phase or the
game over phase).
    /// </summary>
    [HideInInspector]
    public static bool duringGameplay;

    /// <summary>
    /// Set to true if the game is currently paused.
    /// </summary>
    [HideInInspector]
    public static bool gameIsPaused = false;

    [Header("Game Over")]

    /// <summary>
```

```
/// The game over screen default text.
/// </summary>
public GameObject gameoverDefaultText;

/// <summary>
/// The "YOU WON" image displayed when the user wins.
/// </summary>
public GameObject youWonImage;

/// <summary>
/// The "YOU LOST" image displayed when the user wins.
/// </summary>
public GameObject youLostImage;

/// <summary>
/// The game over text of the number of laps the user made.
/// </summary>
public TextMeshProUGUI userLapsMade;

/// <summary>
/// The game over text of the number of laps the AI made.
/// </summary>
public TextMeshProUGUI AILapsMade;


/// <summary>
/// The game over text of the total time of the race.
/// </summary>
public TextMeshProUGUI totalTime;

/// <summary>
/// The game over quit button.
/// </summary>
public GameObject quitButton;

/// <summary>
/// A reference to the audio manager.
/// </summary>
[Space(10)]
public AudioManager audioManager;

/// <summary>
/// Start is called before the first frame update
```

```
/// </summary>
void Start()
{
    audioManager = FindObjectOfType<AudioManager>();
}

/// <summary>
/// Update the lap count of a given agent.
/// </summary>
/// <param name="carAgent">The agent to update it's lap count.</param>
public void UpdateLapCount(CarAgent carAgent)
{
    if (carAgent.isUser)
    {
        userLapCount.text = string.Format("LAP {0}/2", carAgent.LapsMade);
    }

    else
    {
        AILapCount.text = string.Format("LAP {0}/2", carAgent.LapsMade);
    }
}

/// <summary>
/// Display the game over screen.
/// </summary>
/// <returns>
/// Waits before displaying stats.
/// </returns>
/// <param name="user">The user agent.</param>
/// <param name="AI">The AI agent.</param>
public IEnumerator DisplayGameoverScreen(CarAgent user, CarAgent AI)
{
    duringGameplay = false;

    // Wait 2 seconds before displaying the dark screen and the default game over text
    yield return new WaitForSeconds(2);
    darkPanel.SetActive(true);
    gameoverDefaultText.SetActive(true);

    // Display the correct image according to the user winning status
    if (user.LapsMade > AI.LapsMade) youWonImage.SetActive(true);
    else youLostImage.SetActive(true);
```

```
    // Wait 1 second before displaying the game stat
    yield return new WaitForSeconds(1);
    totalTime.text = raceTimer.GetTotalTime();
    totalTime.gameObject.SetActive(true);

    yield return new WaitForSeconds(1);
    userLapsMade.text = string.Format("{0}/2", user.LapsMade);
    userLapsMade.gameObject.SetActive(true);

    yield return new WaitForSeconds(1);
    AILapsMade.text = string.Format("{0}/2", AI.LapsMade);
    AILapsMade.gameObject.SetActive(true);

    // Wait 1 second and display the quit button
    yield return new WaitForSeconds(1);
    quitButton.SetActive(true);
}

/// <summary>
/// Update is called once per frame.
/// </summary>
void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape) && duringGameplay)
    {
        if (gameIsPaused) Resume();

        else
        {
            Pause();
        }
    }
}

public void Resume()
{
    pauseMenu.SetActive(false);
    darkPanel.SetActive(false);
    Time.timeScale = 1f;
    gameIsPaused = false;
}
```

```
    public void Pause()
    {
        darkPanel.SetActive(true);
        pauseMenu.SetActive(true);
        Time.timeScale = 0;
        gameIsPaused = true;
    }

    /// <summary>
    /// Quit the application when pressing on the quit button.
    /// </summary>
    public void OnQuitButtonClick()
    {
        Application.Quit();
    }

    public void MuteAudio()
    {
        if (AudioManager.audioIsMuted)
        {
            audioManager.SetVolume("MainTheme", 0.5f);
            AudioManager.audioIsMuted = false;
            muteButtonText.text = "MUTE";
        }

        else
        {
            audioManager.SetVolume("MainTheme", 0);
            AudioManager.audioIsMuted = true;
            muteButtonText.text = "UNMUTE";
        }
    }
}
```

SceneLoader.cs

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

/// <summary>
/// A helper class for loading scenes asynchronously and displaying a loading bar.
/// </summary>
public class SceneLoader : MonoBehaviour
{
    /// <summary>
    /// The screen for the loading process.
    /// </summary>
    public GameObject loadingScreen;

    /// <summary>
    /// The loading bar slider.
    /// </summary>
    public Slider progressSlider;

    /// <summary>
    /// Load a given scene.
    /// </summary>
    /// <param name="sceneName">The scene to load.</param>
    public void LoadScene(string sceneName)
    {
        StartCoroutine(LoadAsync(sceneName));
    }

    /// <summary>
    /// Load a given scene asynchronously and display a loading bar;
    /// </summary>
    /// <returns>
    /// Waits for the loading to finish.
    /// </returns>
    /// <param name="sceneName">The scene to load.</param>
    IEnumerator LoadAsync (string sceneName)
    {
        // Load the scene asynchronously
        AsyncOperation sceneLoadingOperation =
SceneManager.LoadSceneAsync(sceneName);
```

```
        // While the scene is loading, if the scene is either the racetrack or the parking
mode,
        // display the loading progress bar.
        if (sceneName != "ChooseMode" && sceneName != "MainMenu")
        {
            loadingScreen.SetActive(true);

            while (!sceneLoadingOperation.isDone)
            {
                float loadingProgress = Mathf.Clamp01(sceneLoadingOperation.progress /
0.9f);
                progressSlider.value = loadingProgress;
                yield return null;
            }
        }
    }
}
```

Sound.cs

```
using UnityEngine;

/// <summary>
/// A class representing an audio file
/// </summary>
[System.Serializable]
public class Sound
{
    /// <summary>
    /// The name of the track
    /// </summary>
    public string name;

    /// <summary>
    /// The AudioClip of the track
    /// </summary>
    public AudioClip clip;

    /// <summary>
    /// The volume of the track (goes between 0 and 1).
    /// </summary>
```

```
    [Range(0f, 1f)]
    public float volume;

    /// <summary>
    /// The pitch of the track (goes between 0.1 and 3).
    /// </summary>
    [Range(0.1f, 3f)]
    public float pitch;

    /// <summary>
    /// Whether or not the track should be looped.
    /// </summary>
    public bool loop;

    /// <summary>
    /// The AudioSource component of the track.
    /// </summary>
    [HideInInspector]
    public AudioSource source;

}
```

Timer.cs

```
using UnityEngine;
using TMPro;

/// <summary>
/// The basic timer class.
/// This is the base for both the racetrack timer and the parking timer.
/// </summary>
public abstract class Timer : MonoBehaviour
{
    /// <summary>
    /// The timer text displaying the total time of the timer.
    /// </summary>
    public TextMeshProUGUI timerText;

    /// <summary>
    /// the timer panel.
    /// </summary>
    public GameObject timerPanel;
```

```
/// <summary>
/// the total time of the timer.
/// </summary>
public float totalTime;

/// <summary>
/// whether or not the timer is paused.
/// </summary>
protected bool paused;

/// <summary>
/// Pause the timer.
/// </summary>
public void Pause()
{
    paused = true;
}

/// <summary>
/// Show/Hide the timer based on the given activation status.
/// </summary>
/// <param name="activationStatus">Whether on not to show the timer.</param>
public void Show(bool activationStatus)
{
    timerPanel.SetActive(activationStatus);
    timerText.gameObject.SetActive(activationStatus);
}

/// <summary>
/// Insert a space between each character of the timer text.
/// </summary>
/// <returns>
/// The spaced timer.
/// </returns>
/// <param name="time">The timer text to space.</param>
protected string SpaceCharacters(string time)
{
    string spacedTimer = "";

    for (int i = 0; i < time.Length; i++)
    {
        if (i != time.Length - 1) spacedTimer += time[i] + " ";
```

```
            else spacedTimer += time[i];
        }

        return spacedTimer;
    }

    /// <summary>
    /// Convert the timer text into the desired format.
    /// For the racetrack mode- mm:ss:ms.
    /// For the parking mode- ss:ms.
    /// </summary>
    /// <returns>
    /// The time in the desired form.
    /// </returns>
    /// <param name="time">The time to convert.</param>
    public abstract string ConvertToTimerFormat(float time);
}
```