Универзитет у Крагујевцу

Факултет инжењерских наука

Документација за пројекат из предмета:
Основи машинског и дубоког учења

Тема:

Аутономна вожња формула имплементирана као видео игра коришћењем учења подстицајем

Студенти:
Михаило Максимовић 602/2018
Ђорђе Гачић 626/2018

Професор:
Др Владимир Миловановић
Асистент:
Никола Радовановић

Крагујевац, фебруар 2022.

Садржај:

1 Увод

Машинско учење је подобласт вештачке интелигенције чији је циљ конструисање алгоритама и рачунарских система који су способни да се адаптирају на нове ситуације и уче на бази искуства. Развијене су различите технике учења за извршавање различитих задатака. У овоме пројекту биће разматрана техника учења подстицајем (енг. Reinforcement Learning) кроз имплементацију у видео игри трке формула. Учење подстицајем је област машинског учења која се бави изучавањем начина на који интелигентни агенти треба да предузму акције у окружењу како би максимизирали појам кумулативне награде.

Учење подстицајем је једна од три основне парадигме машинског учења, поред надгледаног и ненадгледаног учења. Изучавањем технике учења подстицајем развили су се многи занимљиви и корисни пројекти тако да је ова техника постала веома популарна и ушла у широку употребу.

Мотивација за израду овог пројекта била је идеја аутономне вожње. Кроз трку болида који самостално возе, на неком вишем нивоу од овог пројекта, могу се организовати такмичења научених модела што може бити веома занимљиво ако би се спровело у пракси. У наставку ће бити објашњене тахнологије коришћене при изради пројекта као и начин на који је пројекат имплементиран.

Идеја пројекта је да се особа која игра игру и управља једном формулом утркује са осталим формулама које представљају утрениране моделе.

2 Опис коришћене технологије

2.1 Списак коришћене технологије

За израду пројекта коришћене су следеће технологије:

- Оперативни систем: Windows 10.
- Unity - мулти-платформски софтвер за израду игара који данас подржава развој на десктоп, мобилним и конзолним платформама, као и на платформама за виртуелну реалност.
- ML-Agents - пројекат отвореног кода који омогућава играма и симулацијама да служе као окружења за обуку интелигентних агената. Пружа имплементације најсавременијих алгоритама (засновани на PyTorch-у) како би се омогућило програмерима игара да лако обуче интелигентне агенте за своје игре.
- Python 3.7.9 - потребан да би се исправно учитале библиотеке потребне за тренирање модела и да би се преузео поменути PyTorch који представља библиотеку отвореног кода намењену за машинско учење.
- C# - програмски језик који користи Unity за функционалност модела, а самим тим и за писање кода потребног за тренирање модела коришћењем учења подстицајем
- Latex - за израду документације.
- www.overleaf.com - онлајн Latex компајлер.
- Google Slides - за израду презентације.

2.2 Инсталација и подешавање окружења

Пре свега је потребно инсталирати Unity софтвер са следеће, а могуће га је преузети преко следећег линка.

Прво је потребно преузети Unity-hub који представља неку врсту менаџера наших Unity пројеката и различитих верзија Unity софтвера. Такође, могуће га је инсталирати како на Windows тако и на Linux и MAC оперативним системима.
Након тога је потребно направити нови Unity пројекат.

Следећи корак је подешавање ML-Agents алата за машинско учење. За то је потребана инсталација праве верзије Python-a (коришћена је верзија 3.7.9 и свака наредна верзија до 3.8 укључујући и њу би требало да одговара).

Ако у конзоли тј. командном прозору укуцамо `python --version` требало би да на стандардном излазу испише варзију, што значи да је Python успешно инсталиран.

Потребно је у командном прозору ући у фолдер новокреираног Unity пројекта тако што се укуца `cd <path-to-your-unity-project>`.

Након тога треба креирати виртуелно окружење унутар тог фолдера тако што укуцамо

`python -m venv venv` (или `py -m venv venv` на Windows-у) - друго `venv` представља назив креираног окружења и може бити произвољан. Затим је потребно да се активира виртуелно окружење преко следеће команде: `venv/Scripts/activate` за Windows комадни прозор или `source venv/bin/activate` за Linux конзолу (терминал).

Наредни корак је ажурирање Python-овог `pip` инсталера преко команде: `python -m pip install --upgrade pip` да бисмо преко њега инсталирали PyTorch који садржи библиотеке за машинско учење и без кога ML-Agents не би радио. Ако је ажурирање прошло како треба прелази се на инсталирање PyTorch библиотеке. Инсталира се преко команде:
`pip install torch==1.7.0 -f https://download.pytorch.org/whl/torch_stable.html`

Остао је још корак за инсталацију самог ML-Agents пакета. Ако су претходни кораци успешно прошли ML-Agents се инсталира преко команде `pip install mlagents`. Да бисмо проверили да ли је исправно инсталиран укуцамо команду `mlagents-learn --help` и ако без грешака избацује опције за рад са ML-Agents онда је све спремно за даљу упортебу и креирање пројекта.

Што се тиче Unity пројекта, након свих подешавања потребно је отворити Unity едитор и преузети ML-Agents у оквиру њега. То се ради на следећи начин. Кликне се на `Window > Package Manager`. Отвара се прозор гдје је потребно селектовати `Packages: In Projects` и чекирати `Unity Registry`. Прелистати избор и пронаћи `ML Agents` опцију. Селектовати је и кликнути `Install` у доњем десном углу прозора. Овим је подешавање окружења за машинско учење у оквиру Unity-ја завршено и прелази се на израду окружења, тркачке стазе и модела формуле као и кода за тренирање модела.

3 Изглед графичких компоненти игре

С обзиром да је у овом пројекту акценат на тренирању модела преко учења подстицајем, графичке компоненте су поједностављене како би модел био што једноставнији за тренирање и мање захтеван за окружење.

За модел формуле коришћен је бесплатан Unity Asset (линк) и модел у пројекту има следећи изглед:

Slika 1: Изглед модела формуле

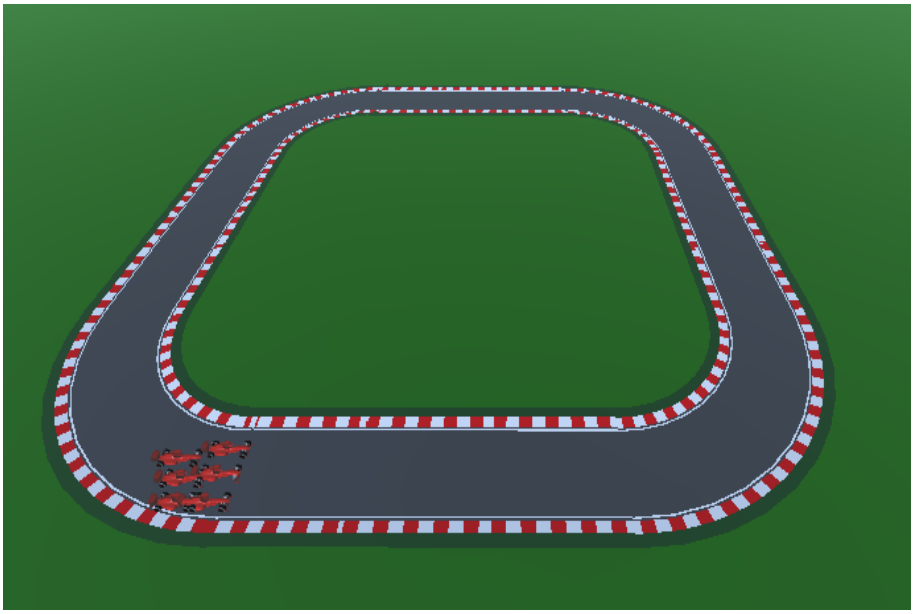
Модел подлоге (терен компонента) има текстуру зелене боје и представља позадину на којој ће касније бити компоненте за стазу.

За модел стазе коришћен је бесплатан Unity Asset (линк) и представља компоненте од којих је направљен жељени изглед стазе.

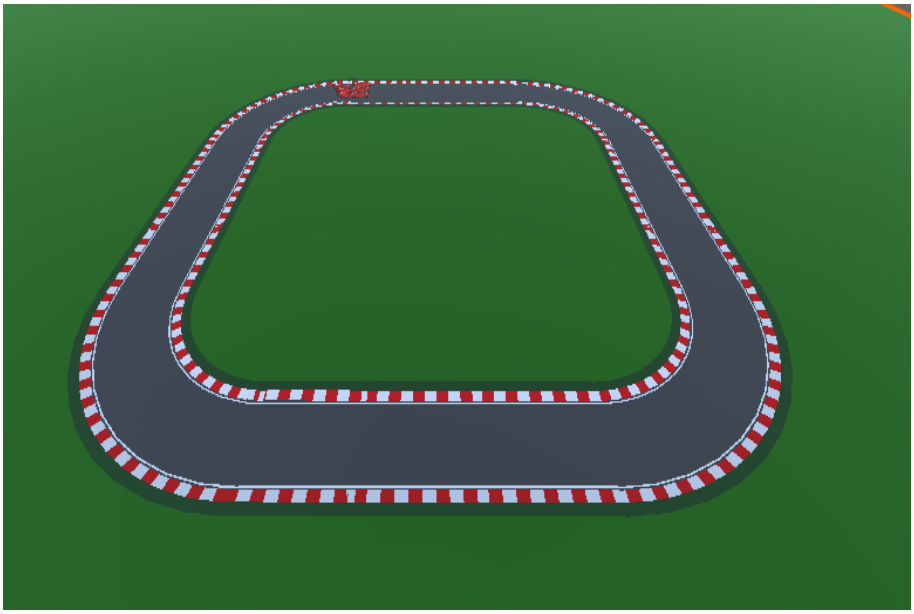
У оквиру обог пројекта постоје три стазе које су одвојене и на којима се тренирају модели:

- Стаза за кретање само на лево
- Стаза за кретање само на десно
- Стаза са комбинованим скретањима на десно или лево - најкомпликованија

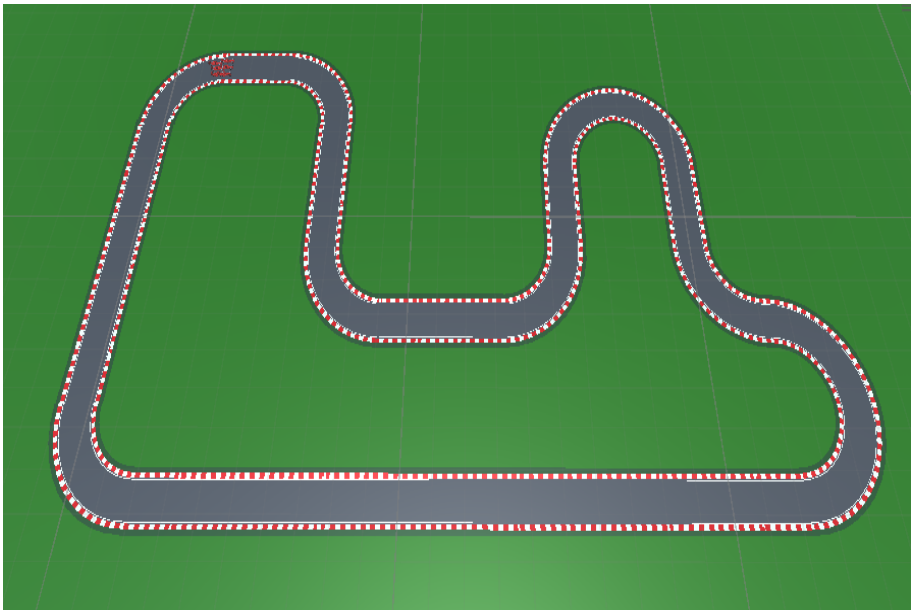
У наставку ће бити илустративно приказане све врсте набројаних стаза редом како су набројане:



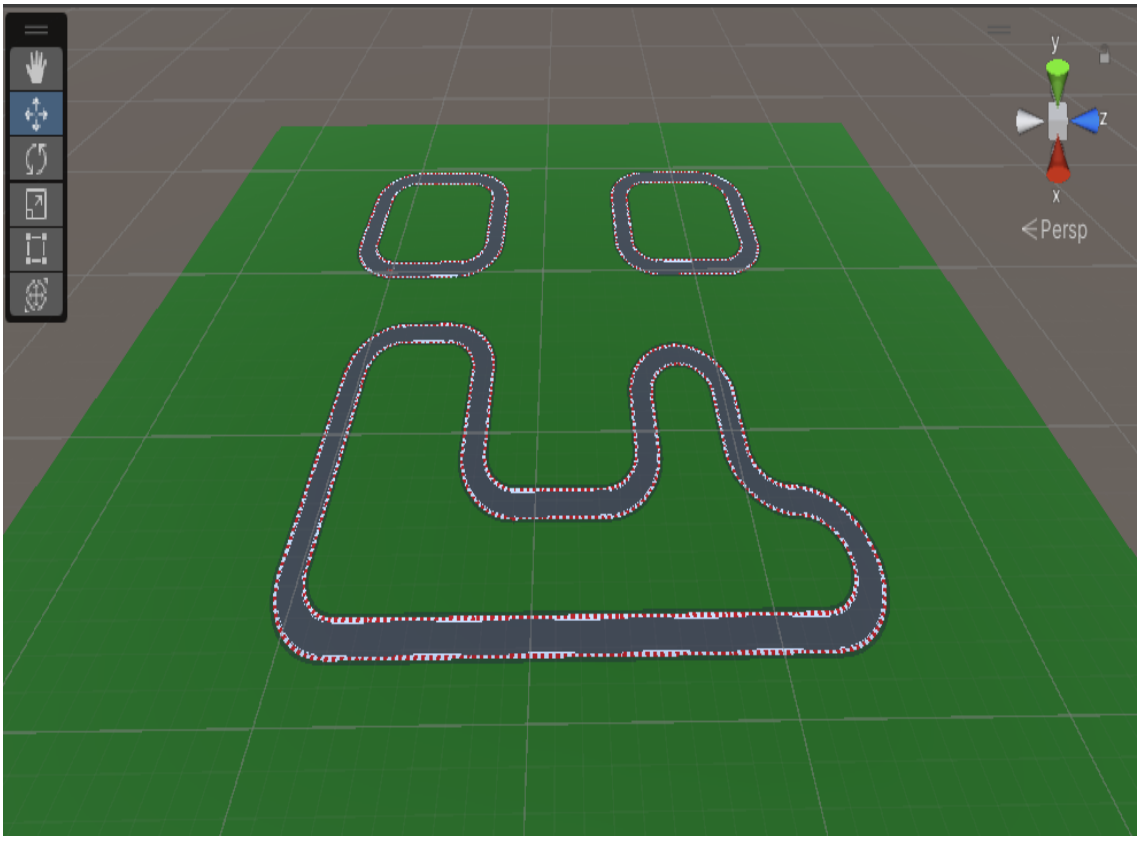
Slika 2: Стаза са скретањима на лево



Slika 3: Стаза са скретањима на десно

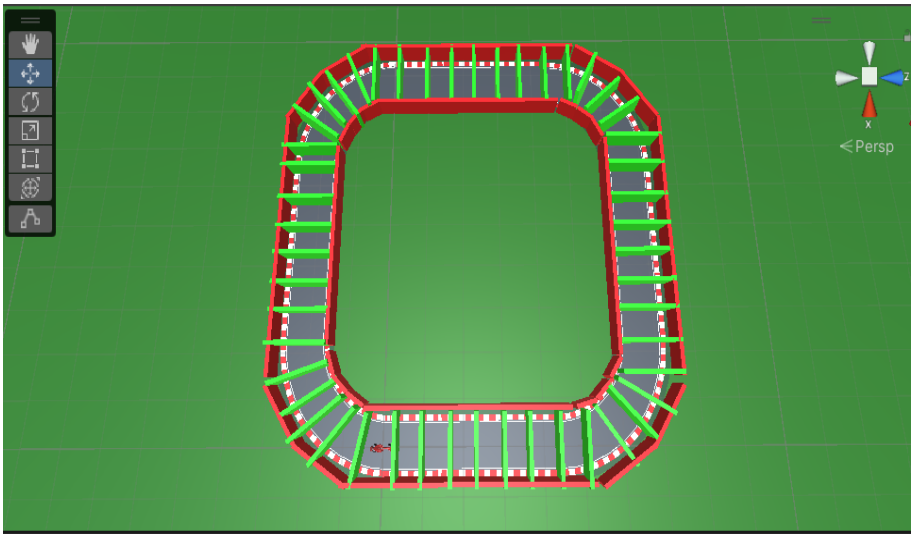


Slika 4: Стаза са комбинованим скретањима

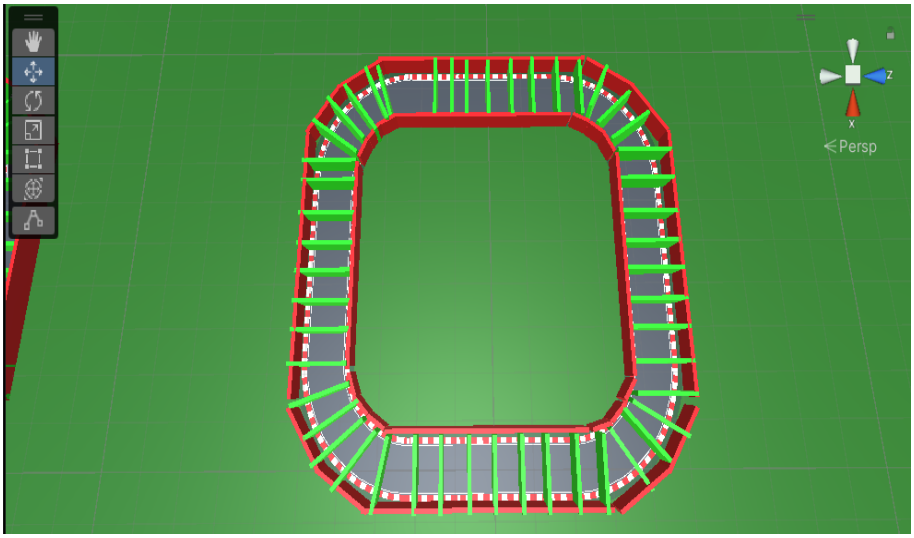


Slika 5: Све стазе у једном приказу

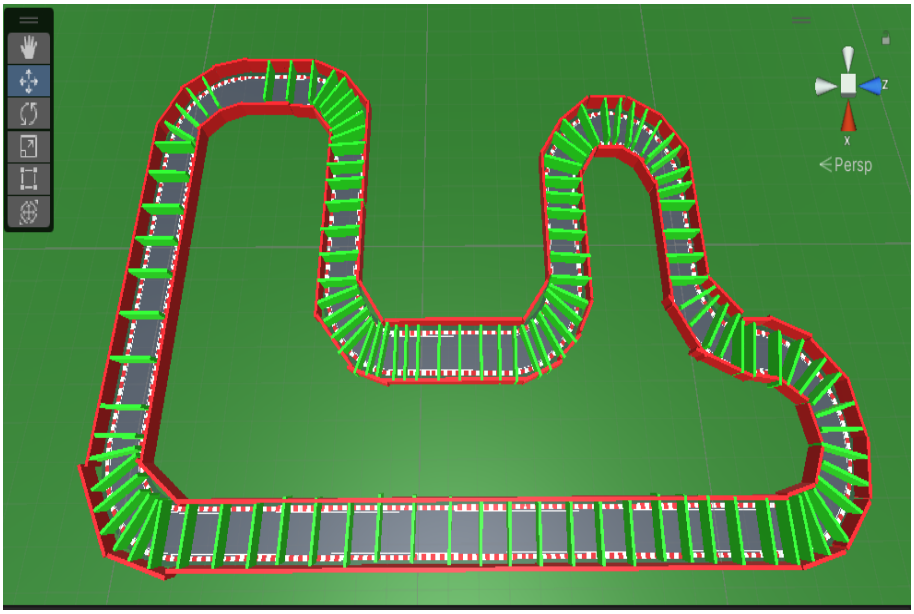
Оно што је изостављено на претходним сликама, а што ће бити приказано у наставку јесу исте стазе али са приказаним зидовима (Walls) и контролним тачкама (Checkpoints) који су након покретања невидљиви али се виде приликом израде пројекта. Они служе за то да би се формула могла наградити уколико додирне исправан checkpoint или казнити (наградити негативно) уколико додирне зид или погрешну контролну тачку (Checkpoint).



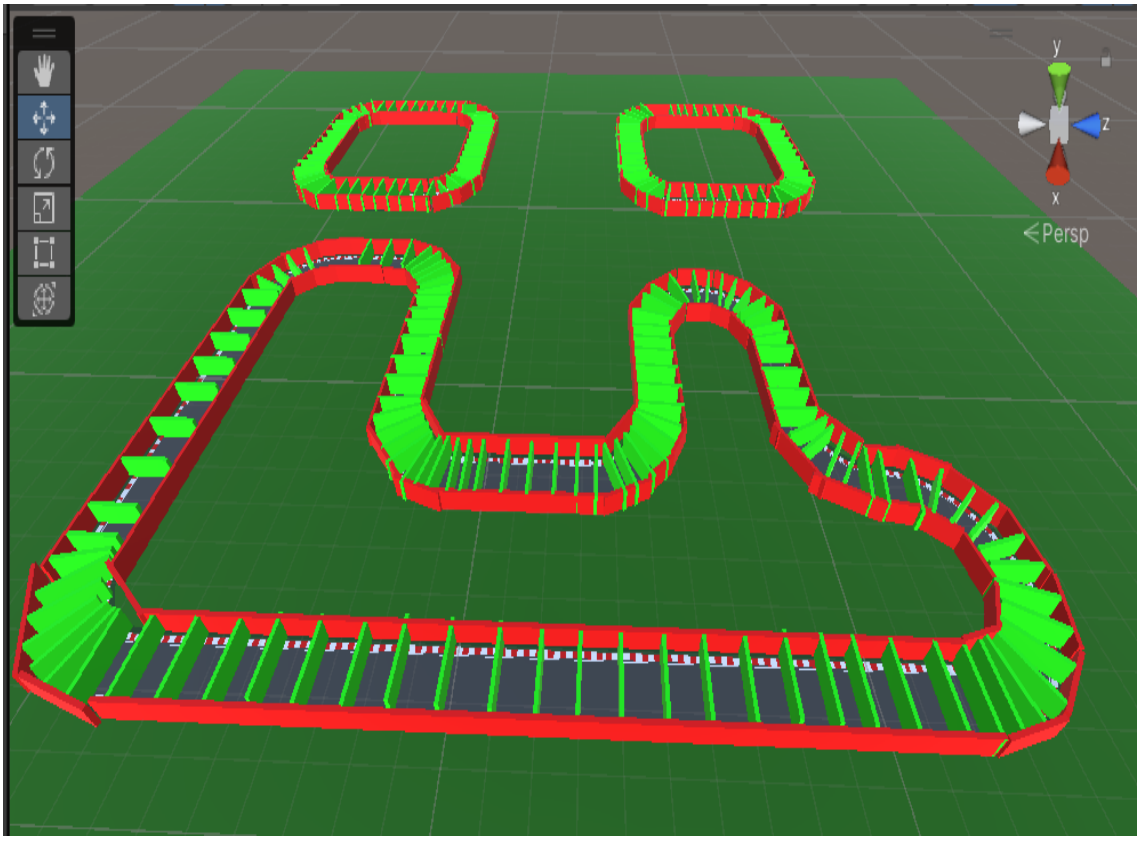
Slika 6: Стаза лево + зидови и checkpoints



Slika 7: Стаза десно + зидови и checkpoints



Slika 8: Велика стаза + зидови и checkpoints



Slika 9: Све стазе + зидови и checkpoints

4 Опис функционалности модела формуле

Функционалност модела формуле је дефинисана кроз датотеку carDriver.cs и carDriverPlayer.cs (C# скрипте) чији ће садржај бити објашњен у наставку. Треба напоменути да се у овим датотекама не налази имплементација за учење подстицајем. Њихов садржај ће бити накнадно објашњен у секцији за опис имплементације учења подстицајем.

Опис садржаја датотеке `carDriver.cs`:
- садржи класу `carDriver` у којој се дефинишу следеће променљиве (за брзину, убрзање, кочење, угаону брзину, ... као и RigidBody променљива која омогућује објекту да реагује на физику у реалном времену):

carDriver.cs

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class carDriver : MonoBehaviour
6   {
7       private const int SOLID_OBJECTS_LAYER = -1;
8
9       private float speed;
10      private float speedMax = 30f;
11      private float speedMin = -50f;
12      private float acceleration = 10f;
13      private float brakeSpeed = 100f;
14      private float reverseSpeed = 20f;
15      private float idleSlowdown = 10f;
16
17      private float turnSpeed;
18      private float turnSpeedMax = 300f;
19      private float turnSpeedAcceleration = 300f;
20      private float turnIdleSlowdown = 500f;
21
22      private float forwardAmount;
23      private float turnAmount;
24
25      private Rigidbody carRigidbody;
```

Slika 10: Дефиниција променљивих у carDriver класи

Класа carDriver садржи следеће функције:

- `private void Awake()` - инстанцира се променљива `carRigidBody`
- `private void FixedUpdate()` - позива се при сваком фиксном фрејму и садржи највећи део понашања самог објекта формуле
- `private void OnCollisionEnter(Collision collision)` - региструје колизију са другим објектом

- `public void SetInputs(float forwardAmount, float turnAmount)` - позива се када играч управља објектом формуле
- `public void ClearTurnSpeed()` - подешава угаону брзину на нула
- `public float GetSpeed()` - враћа тренутну брзину
- `public void SetSpeedMax(float speedMax)` - подешава максималну брзину
- `public void SetTurnSpeedMax(float turnSpeedMax)` - подешава максималну угаону брзину
- `public void SetTurnSpeedAcceleration(float turnSpeedAcceleration)` - подешава угаоно убрзање
- `public void StopCompletely()` - зауставља кретање модела постављањен брзине и угаоне брзине на нула
- `public Vector3 GetRigidbodyVelocity()` - враћа брзину (velocity) RigidBody објекта

Опис садржаја датотеке `carDriverPlayer.cs`:
- садржи класу `carDriverPlayer` у којој се дефинише променљива типа carDriver. Класа `carDriverPlayer` омогућује да се од играча региструју команде и проследе објекту `carDriver` класе. Саджај ове класе ће бити приказан у потпуности на наредној слици јер је имплементација кратка:

carDriverPlayer.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class carDriverPlayer : MonoBehaviour
6  {
7      // Start is called before the first frame update
8      private carDriver carDriver;
9
10     private void Awake()
11     {
12         carDriver = GetComponent<carDriver>();
13
14     }
15
16     private void Update()
17     {
18         float forwardAmount = Input.GetAxisRaw("Vertical");
19         float turnAmount = Input.GetAxisRaw("Horizontal");
20         carDriver.SetInputs(forwardAmount, turnAmount);
21     }
22 }
23
```

Slika 11: Садржај carDriverPlayer.cs датотеке

Класа `carDriverPlayer` садржи следеће функције:

- `private void Awake()` - инстанцира се променљива `carDriver`
- `private void Update()` - позива се при сваком фрејму и региструје команде (улазе) које задаје играч и прослеђује их `SetInputs` функцији `carDriver` објекта.

5 Опис имплементације учења подстицајем

Учење подстицајем (Reinforcement Learning) се састоји од цикличног процеса разматрања, одлуке, акције и награде као што је то приказано на слици испод:

Slika 12: Процес учења подстицајем

Кроз све ове процесе потребно је да прође модел који тренирамо како би исправно научио да заврши стазу.

Имплементација везана за тренирање модела садржана је у следеће три датотеке:

- `CheckpointSingle.cs` - имплементација за једну контролну тачку
- `TrackCheckpoints.cs` - имплементација за све контролне тачке на стази
- `carDriverAgent.cs` - имплементација агента - формуле (модела који учи)

Да бисмо наставили са објашњавањем техника коришћених при имплементацији алгоритма машинског учења, прво је потребно да размотримо принцип контролних тачака и разлог због којих оне постоје. Наиме, постојање контролних тачака на одређеном растојању по стази прикупља информације о томе где се модел који учи тренутно налази и на основу тога му даје одређену награду било позитивну било негативну. Циљ модела је да максимизира награду и на основу тога се заправо одвија његово учење. Алгоритам за сваки покренути модел води рачуна која је једина исправна контролна тачка кроз коју модел треба проћи како би добио награду и ако заиста и прође кроз њу биће награђен и то ће се сматрати као напредак. Такође, негативна награда постоји и ако модел удари

у зид тј. ако скрене са стазе.

Опис садржаја датотеке `CheckpointSingle.cs`:
- садржи класу `CheckpointSingle` у којој се дефинише променљива типа `TrackCheckpoints`.
Саджај ове класе ће бити приказан у потпуности на наредној слици јер је имплементација кратка:

CheckpointSingle.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class CheckpointSingle : MonoBehaviour
6  {
7      public TrackCheckpoints trackCheckpoints;
8      private void OnTriggerEnter(Collider other)
9      {
10         if(other.tag == "Agent") {
11             trackCheckpoints.PlayerThroughCheckpoint(this, other.transform);
12         }
13     }
14
15     public void SetTrackCheckpoint(TrackCheckpoints trackCheckpoints)
16     {
17         this.trackCheckpoints = trackCheckpoints;
18     }
19 }
20
```

Slika 13: Садржај CheckpointSingle.cs датотеке

Класа `CheckpointSingle` садржи следеће функције:

- `private void OnTriggerEnter(Collider other)` - региструје ако агент додирне контролну тачку и прослеђује ту информацију и податке о агенту објекту `TrackCheckpoints` преко његове функције `PlayerThroughCheckpoint`
- `public void SetTrackCheckpoint(TrackCheckpoints trackCheckpoints)` - служи за иницијализацију променљиве типа `TrackCheckpoints`

Опис садржаја датотеке `TrackCheckpoints.cs`:
Класа `TrackCheckpoints` садржи следећа поља:

- `public event EventHandler<CarCheckpointEventArgs> OnCarCorrectCheckpoint` - слушалац догађаја дохватања исправне контролне тачке
- `public event EventHandler<CarCheckpointEventArgs> OnCarWrongCheckpoint` - слушалац догађаја дохватања погрешне контролне тачке

- `[SerializeField] public List<Transform> carTransformList` - листа података о свим агентима који уче
- `private List<int> nextCheckpointSingleIndexList` - листа индекса следеће исправне контролне тачке коју треба да дохвати сваки од агената агент
- `private List<CheckpointSingle> checkpointSinglesList` - листа појединачних контролних тачака на стази
- класа `public class CarCheckpointEventArgs` која наслеђује `EventArgs` и садржи само једно поље које представља податке о агенту који је додирнуо контролну тачку

Класа `TrackCheckpoints` садржи следеће функције:

- `private void Awake()` - проналази све контролне тачке на стази и смешта их у низ свих појединачних контролних тачака и поставља први следећи исправни индекс сваког агента на нула. Представља почетну конфигурацију.
- `public void PlayerThroughCheckpoint(CheckpointSingle cps, Transform tr)` - рукује догађајем када неки од агената (формула које уче) дохвати неку од контролних тачки
- `public void ResetCheckpoint(Transform carTransform)` - ресетује следећу исправну контролну тачку за агента чији се подаци прослеђују као аргумент
- `public CheckpointSingle GetNextCheckpoint(Transform carTransform)` - дохвата следећу исправну контролну тачку за агента чији се подаци прослеђују као аргумент

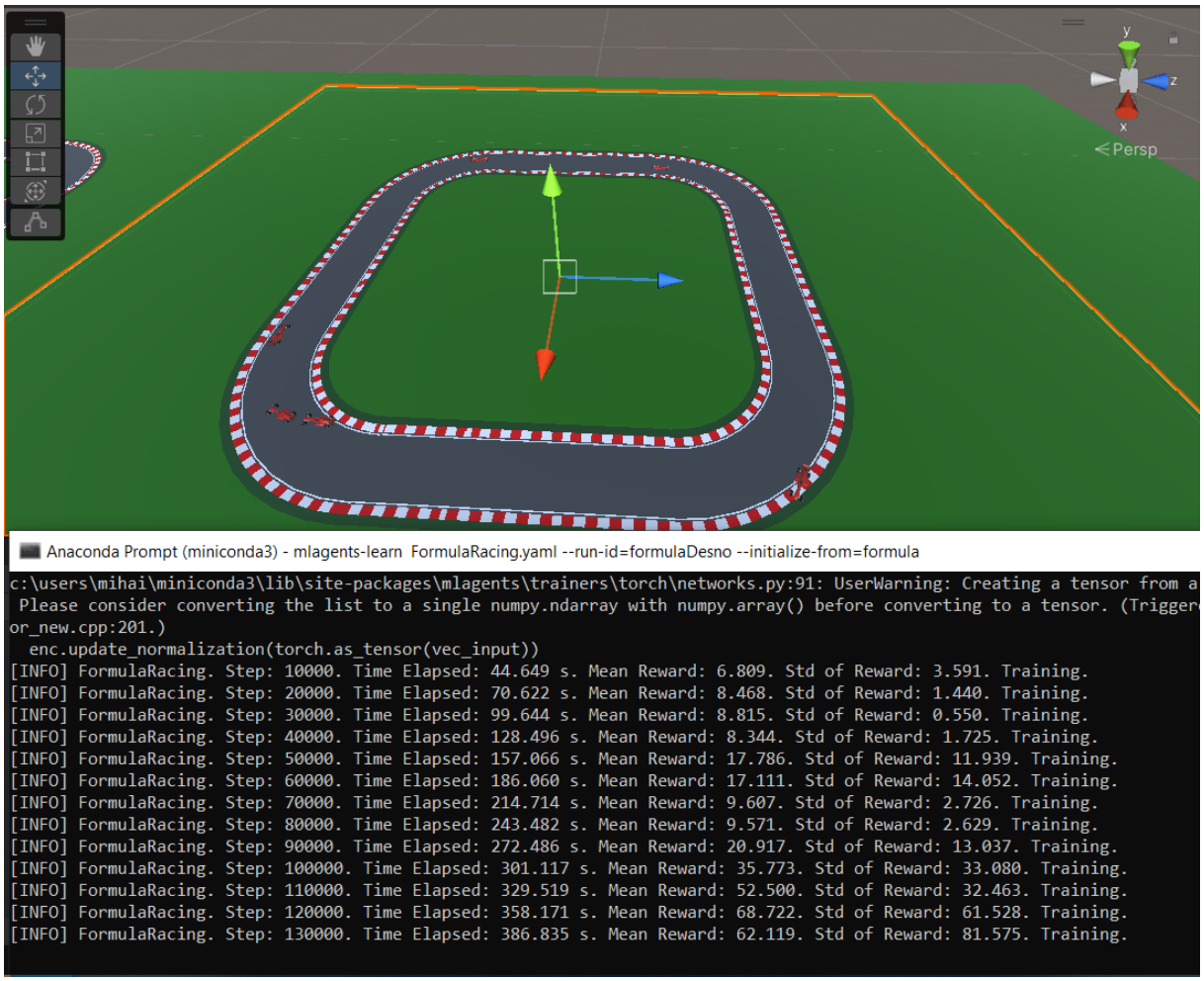
Опис садржаја датотеке `carDriverAgent.cs`:
- садржи `carDriverAgent` класу која наслеђује класу `Agent` из ML-Agents пакета Класа `carDriverAgent` садржи следећа поља:

- `[SerializeField] public TrackCheckpoints trackCheckpoints` - податак о свим контролним тачкама на стази
- `[SerializeField] public Transform spawnPosition` - садржи податке о положају агента приликом рестартовања
- `public event EventHandler OnCarCorrectCheckpoint` - слушалац догађаја за пролазак кроз исправну контролну тачку
- `public event EventHandler OnCarWrongCheckpoint` - слушалац догађаја за пролазак кроз погрешну контролну тачку
- `private carDriver carDriver` - садржи све податке и функционалности обичног агента

Класа `TrackCheckpoints` садржи следеће функције:

- `private void Awake()` - инстанцира се објекат типа `carDriver`
- `private void Start()` - подешавање почетне конфигурације за контролне тачке
- private void TrackCheckpoints_OnCarWrongCheckpoint(object sender, TrackCheckpoints.CarCheckpointEventArgs e) - давање негативне награде ако је агент дохватио погрешну контролну тачку
- private void TrackCheckpoints_OnCarCorrectCheckpoint(object sender, TrackCheckpoints.CarCheckpointEventArgs e) - давање позитивне награде ако је агент дохватио погрешну контролну тачку
- `public override void OnEpisodeBegin()` - дешавање на почетку новог круга учења - ресет на почетну конфигурацију
- `public override void CollectObservations(VectorSensor sensor)` - функција за разматрање у оквиру ML-Agents пакета
- `public override void OnActionReceived(ActionBuffers actions)` - подешавање исхода акције предузетих као резултат донесене одлуке
- `public override void Heuristic(in ActionBuffers actionsOut)` - функција за мануелно управљање од стране играча
- `public void OnTriggerEnter(Collider other)` - детектује додир са зидом и даје негативну награду и завршава круг
- `public void OnTriggerStay(Collider other)` - слично као претходна функција само што не завршава круг

Сам процес учења може се пратити у конзоли тако што се исписује награда коју агенти добијају у сваком кругу тренирајући заједно. Један такав испис приказан је на следећој слици:



Slika 14: Испис у току тренирања модела

Такође, битна компонента самог пројетка јесте .yaml датотека која се у нашем случају назива FormulaRacing.yaml и у њој се налази конфигурација коју у позадини користи алгоритам учења подстицајем. Ту нпр. можемо подесити максимаан број итерација кроз које модел пролази приликом једног учења кретања. Садржај датотеке је приказан на следећој слици:

```yaml
behaviors:
  FormulaRacing:
    trainer_type: ppo
    hyperparameters:
      batch_size: 10
      buffer_size: 100
      learning_rate: 3.0e-4
      beta: 5.0e-4
      epsilon: 0.2
      lambd: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
      beta_schedule: constant
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 1000000
    time_horizon: 64
    summary_freq: 10000
```

Slika 15: Приказ конфигурационог .yaml фајла

Након што модел заврши процес учења генерише се .onnx фајл у коме је садршан "мозак"нашег модела тј. оно што је он у току тренирања научио. Тај "мозак"остаје сачуван и после се може учитати како би се демонстрирало то што је модел научио. Такође се може научени алгоритам на једној стази покренути на сасвим другој како би се провјериле способности тренираног агента на тзв. "непознатом терену".

6 Литература

- https://elektrotehnika.github.io/ml/
- YouTube туторијали за коришћење ML-Agents пакета у оквиру Unity окружења - линк1, линк2
- https://assetstore.unity.com/
- https://stackoverflow.com/
- Интернет