

**Politechnika Wrocławskas
Wydział Informatyki i Telekomunikacji**

Kierunek: **INF-PPT**

Specjalność: -

**PRACA DYPLOMOWA
INŻYNIERSKA**

Bot do komputerowej gry wyścigowej

Kamil Matejuk

Opiekun pracy
Dr Marcin Michalski

Słowa kluczowe: Bot, Uczenie maszynowe, Uczenie przez wzmacnianie

Streszczenie

W poniższej pracy został opisany proces implementacji bota poruszającego się w środowisku gry wyścigowej. Gra pozwala na proceduralne generowanie tras i terenów, co pozwoliło stworzyć nieskończoną ilość środowisk testowych. Do treningu bota wykorzystano metodę uczenia maszynowego przez wzmacnianie. W opracowaniu przeanalizowane zostały różne rodzaje obserwacji, akcji oraz nagród. Finalnie bot został wytrenowany na podstawie danych o odległości pojazdu od ewentualnych przeszkód, symulując metodę używaną na co dzień w pojazdach autonomicznych - tzw. lidar [4].

Abstract

The following thesis describes a process of implementing racing bot, which has been taught to navigate racing track environment. Game allows user to procedurally create track and terrain, as well as play created levels. The bot has been trained using reinforcement learning. This thesis analyzes different observations, action types and reward functions, to finally use distance-based observations. This method is similar to how autonomous vehicles operate in real world - using lidar technology [4].

Spis treści

Wstęp	1
1 Stworzenie gry	3
1.0.1 Cel i funkcjonalności	3
1.0.2 Wykorzystane technologie	3
1.0.3 Stworzenie trasy 2D	3
1.0.4 Stworzenie terenu 3D	5
1.0.5 Przypisanie tekstur	5
1.0.6 Znalezienie najbliższego punktu krzywej Bezier	6
1.0.7 Udoskonalone znalezienie najbliższego punktu krzywej Bezier	6
1.0.8 Porównanie metod	7
1.0.9 Optymalizacja czasu	8
1.0.10 Dodanie obiektów	8
2 Wytrenowanie boty	9
2.0.1 Cel i funkcjonalności	9
2.0.2 Wykorzystane technologie	9
2.0.3 Uczenie przez wzmacnianie	9
2.0.4 Metodyka uczenia	10
2.0.5 Analiza wyników	10
2.0.6 Wybór obserwacji	11
2.0.7 Wybór akcji	14
2.0.8 Zdefiniowanie funkcji oceny	15
2.0.9 Dobranie parametrów	16
2.0.10 gamma	17
2.0.11 lambda	17
2.0.12 buffer size	18
2.0.13 batch size	18
2.0.14 learning rate	19
2.0.15 beta	19
2.0.16 epsilon	20
2.0.17 Dalsze etapy treningu	22
Podsumowanie	23
Bibliografia	25
A Uruchomienie gry	27
A.1 Struktura plików	27
A.2 Uruchomienie gry	27
A.3 Kod źródłowy	27
A.4 Poruszanie się w grze	28
B Dokumentacja techniczna wygenerowana przez Doxygen (ENG)	33

B.1	Namespace Index	33
B.2	Hierarchical Index	34
B.3	Class Index	35
B.4	Namespace Documentation	36
B.4.1	RacingGameBot.Data Namespace Reference	36
B.4.2	RacingGameBot.Editors Namespace Reference	36
B.4.3	RacingGameBot.Menu Namespace Reference	36
B.4.4	RacingGameBot.Play Namespace Reference	36
B.4.5	RacingGameBot.Terrains Namespace Reference	37
B.4.6	RacingGameBot.Tests Namespace Reference	37
B.4.7	RacingGameBot.Utils Namespace Reference	37
B.5	Class Documentation	38
B.5.1	RacingGameBot.Data.TerrainGenData Class Reference	38
B.5.2	RacingGameBot.Data.UpdatableData Class Reference	39
B.6	RacingGameBot.Editors.BuildScript Class Reference	39
B.6.1	Member Function Documentation	39
B.6.2	RacingGameBot.Editors.TerrainGeneratorEditor Class Reference	40
B.6.3	RacingGameBot.Editors.TerrainLoaderEditor Class Reference	40
B.6.4	RacingGameBot.Editors.UpdatableDataEditor Class Reference	41
B.6.5	RacingGameBot.Menu.ButtonActions Class Reference	41
B.6.6	Member Function Documentation	42
B.6.7	RacingGameBot.Menu.CreateLevelMenu Class Reference	42
B.6.8	Member Function Documentation	43
B.6.9	RacingGameBot.Menu.EndMenu Class Reference	45
B.6.10	Member Function Documentation	45
B.6.11	RacingGameBot.Menu.MainMenu Class Reference	46
B.6.12	Member Function Documentation	46
B.6.13	RacingGameBot.Menu.PauseMenu Class Reference	48
B.6.14	Member Function Documentation	48
B.6.15	RacingGameBot.Play.CarAgent Class Reference	49
B.6.16	Member Function Documentation	49
B.6.17	RacingGameBot.Play.MultiTraining Class Reference	51
B.6.18	RacingGameBot.Terrains.Loop Class Reference	51
B.6.19	Constructor & Destructor Documentation	52
B.6.20	Member Function Documentation	53
B.6.21	RacingGameBot.Terrains.OrientedPoint Class Reference	54
B.6.22	RacingGameBot.Terrains.TerrainGenerator Class Reference	55
B.6.23	Member Function Documentation	56
B.6.24	RacingGameBot.Terrains.TerrainLoader Class Reference	58
B.6.25	Member Function Documentation	59
B.6.26	RacingGameBot.Tests.BezierTests Class Reference	60
B.6.27	RacingGameBot.Tests.TextureTests Class Reference	61
B.7	RacingGameBot.Utils.Bezier Class Reference	61
B.7.1	Member Function Documentation	62
B.8	RacingGameBot.Utils.Logging Class Reference	64
B.8.1	Member Function Documentation	64
B.8.2	RacingGameBot.Utils.Objects Class Reference	65
B.8.3	Member Function Documentation	65
B.9	RacingGameBot.Utils.Parser Class Reference	67
B.9.1	Member Function Documentation	67



Spis rysunków

1.1	Generowanie punktów kontrolnych krzywej Bezier	5
1.2	Podział terenu w celu optymalizacji ilości obliczeń	8
1.3	Wygenerowane tereny	8
2.1	Prowadzenie wielu treningów na raz	10
2.2	Przykładowa analiza wyników	10
2.3	Obserwacje bota dystansów na około pojazdu	11
2.4	Obserwacje bota kolorów na około pojazdu	12
2.5	Obserwacje bota z kamery przedniej	12
2.6	Obserwacje bota z kamery z lotu ptaka	13
2.7	Porównanie metod obserwacji	13
2.8	Porównanie metod podejmowania akcji	14
2.9	Wizualizacja oceny akcji	15
2.10	Rozmieszczenie punktów kontrolnych	16
2.11	Wybór hiperparametrów: gamma	17
2.12	Wybór hiperparametrów: lambd	17
2.13	Wybór hiperparametrów: buffer size	18
2.14	Wybór hiperparametrów: batch size	18
2.15	Wybór hiperparametrów: learning rate	19
2.16	Wybór hiperparametrów: beta	19
2.17	Wybór hiperparametrów: epsilon	20
2.18	Wybór hiperparametrów: porównanie wyników	20
2.19	Postęp treningu: etap 1	22
A.1	Instrukcja grania - ekran główny	28
A.2	Instrukcja grania - wybór poziomu	29
A.3	Instrukcja grania - gra	30
A.4	Instrukcja grania - zatrzymanie gry	30
A.5	Instrukcja grania - tworzenie poziomu	31
B.1	Inheritance diagram for RacingGameBot.Data.TerrainGenData	38
B.2	Collaboration diagram for RacingGameBot.Data.TerrainGenData	38
B.3	Inheritance diagram for RacingGameBot.Data.UpdatableData	39
B.4	Collaboration diagram for RacingGameBot.Data.UpdatableData	39
B.5	Inheritance diagram for RacingGameBot.Editors.TerrainGeneratorEditor	40
B.6	Collaboration diagram for RacingGameBot.Editors.TerrainGeneratorEditor	40
B.7	Inheritance diagram for RacingGameBot.Editors.TerrainLoaderEditor	40
B.8	Collaboration diagram for RacingGameBot.Editors.TerrainLoaderEditor	40
B.9	Inheritance diagram for RacingGameBot.Editors.UpdatableDataEditor	41
B.10	Collaboration diagram for RacingGameBot.Editors.UpdatableDataEditor	41
B.11	Inheritance diagram for RacingGameBot.Menu.ButtonActions	41
B.12	Collaboration diagram for RacingGameBot.Menu.ButtonActions	41
B.13	Inheritance diagram for RacingGameBot.Menu.CreateLevelMenu	42
B.14	Collaboration diagram for RacingGameBot.Menu.CreateLevelMenu	42
B.15	Inheritance diagram for RacingGameBot.Menu.EndMenu	45

B.16 Collaboration diagram for RacingGameBot.Menu.EndMenu	45
B.17 Inheritance diagram for RacingGameBot.Menu.MainMenu	46
B.18 Collaboration diagram for RacingGameBot.Menu.MainMenu	46
B.19 Inheritance diagram for RacingGameBot.Menu.PauseMenu	48
B.20 Collaboration diagram for RacingGameBot.Menu.PauseMenu	48
B.21 Inheritance diagram for RacingGameBot.Play.CarAgent	49
B.22 Collaboration diagram for RacingGameBot.Play.CarAgent	49
B.23 Inheritance diagram for RacingGameBot.Play.MultiTraining	51
B.24 Collaboration diagram for RacingGameBot.Play.MultiTraining	51
B.25 Inheritance diagram for RacingGameBot.Terrains.TerrainGenerator	55
B.26 Collaboration diagram for RacingGameBot.Terrains.TerrainGenerator	55
B.27 Inheritance diagram for RacingGameBot.Terrains.TerrainLoader	58
B.28 Collaboration diagram for RacingGameBot.Terrains.TerrainLoader	58
B.29 Inheritance diagram for RacingGameBot.Tests.BezierTests	60
B.30 Collaboration diagram for RacingGameBot.Tests.BezierTests	60
B.31 Inheritance diagram for RacingGameBot.Tests.TextureTests	61
B.32 Collaboration diagram for RacingGameBot.Tests.TextureTests	61
B.33 Inheritance diagram for RacingGameBot.Utils.Objects	65
B.34 Collaboration diagram for RacingGameBot.Utils.Objects	65

Spis tabelic

1.1 Porównanie metod rzutowania punktu na krzywą Bezier	7
2.1 Wybór hiperparametrów: porównanie wyników	20

Wstęp

W obecnych czasach obserwujemy coraz bardziej dynamiczny rozwój branży pojazdów wyposażonych w sterowanie autonomiczne. Autonomia pojazdu określana jest w skali pięciostopniowej, gdzie stopień 0 oznacza brak autonomii, a stopień 5 - pełną automatyzację kierowania pojazdem [6]. Aktualnie, jako ludzkość, jesteśmy w stanie osiągnąć autonomię poziomu 2, co oznacza, że kierowca musi nieustannie czuwać nad decyzjami samochodu, a software ma prawo nie rozumieć skrzyżowań bez sygnalizacji świetlnej czy pojazdów uprzywilejowanych.

Oprogramowanie takich pojazdów często testowane jest w symulowanym i bezpiecznym środowisku, które jednak twórcy samochodów starają się jak najbardziej upodabniać do świata realnego. Pojazdy te wykorzystują m.in. sensory lidar[4] oraz kamery jako główne źródło informacji. Następnie na podstawie tych danych i z wykorzystaniem algorytmów uczenia maszynowego poddawane są treningowi.

W poniżej pracy zostało zasymulowane środowisko naturalne oraz pojazd, wraz z zachowanymi w znacznym stopniu prawami fizyki oraz sposobem sterowania. Metody obserwacji otoczenia inspirowane były metodami wykorzystywanyimi w rzeczywistości. Pojazd został wytrenowany do poruszania się po torze w różnych środowiskach.

W pierwszym rozdziale przedstawiony został proces tworzenia gry jako proceduralnego środowiska do testów. Kolejny rozdział zawiera informacje o wyborze optymalnych parametrów ze względu na rozmiar sieci, czas uczenia oraz inferencji. Finalnie przedstawiony został proces trenowania bota.



Rozdział 1

Stworzenie gry

1.0.1 Cel i funkcjonalności

Gra została stworzona przede wszystkim jako środowisko treningowe oraz testowe bota, dopiero w drugiej kolejności jako rozrywka dla użytkownika. Dostarczona w dodatku *A* gra pozwala graczowi na załadowanie jednego z domyślnych terenów, bądź stworzenie własnego terenu i przetestowanie go w grze. Celem rozgrywki jest przetransportowanie pojazdu od miejsca startowego do mety, nie zjeżdżając ze ścieżki. Gra kończy się po wykonaniu jednego pełnego okrążenia. Gracz konkuuruje z 3 innymi pojazdami, sterowanymi przez wytrenowanego bota. Dodatkowo gracz posiada możliwość stworzenia własnego terenu i trasy, sterując kilkoma parametrami. W ten sam sposób zostały przygotowane tereny, na których został wytrenowany bot.

1.0.2 Wykorzystane technologie

Gra została stworzona z wykorzystaniem silnika do gier Unity Engine 3D. Jest to jeden z dwóch najpopularniejszych silników do tworzenia gier, poza Unreal Engine, natomiast posiada on niższą barierę wejścia. Wykorzystanie Unity pozwoliło na zasymulowanie praw fizyki występujących przy prowadzeniu pojazdu.

1.0.3 Stworzenie trasy 2D

Pierwszym etapem generowania terenu jest wybór krzywej opisującej trasę. Każda wygenerowana trasa jest krzywą zamkniętą. Składa się ona z segmentów, których ilość jest bezpośrednio powiązana z zawartością trasy.

Aby wybrać końce segmentów, losowane są punkty $P = (P_x, P_y)$ na płaszczyźnie z uniwersalnym prawdopodobieństwem, z zakresu $P_x \in [-1, 1]$, $P_y \in [-1, 1]$, upewniając się że żadne 2 punkty nie są za blisko siebie.

Algorithm 1.1: Wybór punktów trasy

Data:

- 1 $n \leftarrow$ ilość segmentów;
- 2 $d \leftarrow$ minimalna odległość między punktami;
- 3 $positions \leftarrow []$;
- 4 **while** $positions.Length < n$ **do**
- 5 $pNew \leftarrow new point$;
- 6 $pNew.x \leftarrow random(-1, 1)$;
- 7 $pNew.y \leftarrow random(-1, 1)$;
- 8 $tooClose \leftarrow false$;
- 9 **foreach** $P \in positions$ **do**
- 10 **if** $distance(P, pNew) < d$ **then**
- 11 $tooClose \leftarrow true$;
- 12 **break**;
- 13 **if** $not tooClose$ **then**
- 14 $add pNew.$ to $positions$;

Po wyborze odpowiedniej ilości punktów, sortowane są one zgodnie z ruchem wskazówek zegara, na podstawie kąta pomiędzy wektorem $[0, 1]$, a wektorem kończącym się w danym punkcie.

Algorithm 1.2: Wyznaczenie kąta pomiędzy punktem a prostą OY

Data: $P \leftarrow$ punkt

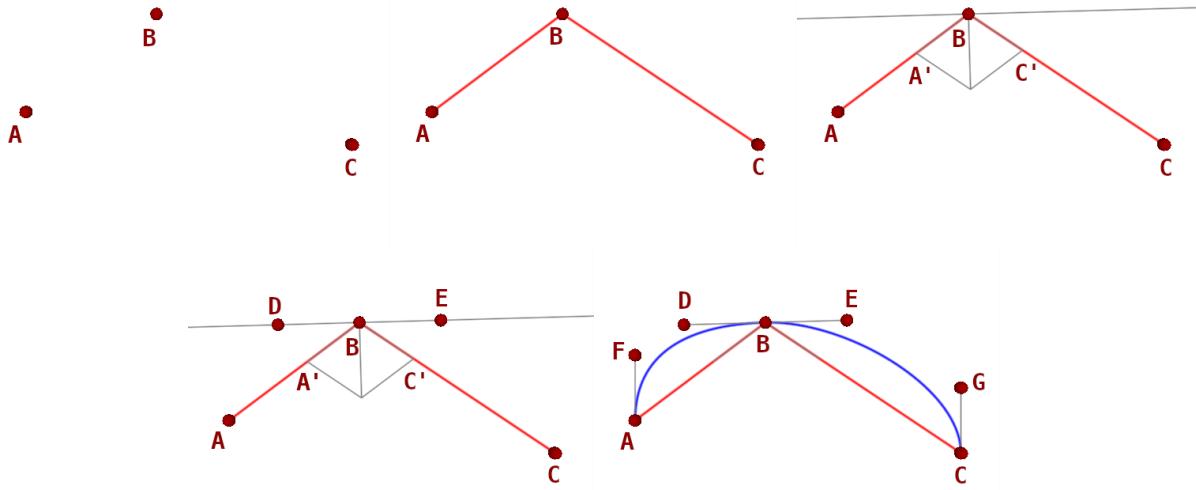
- 1 $forwardVector \leftarrow [0, 1].normalized$;
- 2 $pointVector \leftarrow [P.x, P.y].normalized$;
- 3 $angle \leftarrow \text{acos}(\text{dot}(forwardVector, pointVector))$; /* Arcus cosinus iloczynu skalarnego */
- 4 $sign \leftarrow \text{cross}(forwardVector, pointVector).z$; /* iloczyn wektorowy */
- 5 **if** $sign \geq 0$ **then**
- 6 **return** $angle$;
- 7 **else**
- 8 **return** $-1 \cdot angle$;

Następnie trasa w każdym segmencie wyznaczana jest za pomocą krzywej Bezier 3 stopnia, ponieważ pozwala ona w stosunkowo prosty sposób zapisać stosunkowo komplikowane krzywe, utrzymując płynny kształt. Każda krzywa Bezier składa się z punktu startowego P_0 , końcowego P_3 , oraz dwóch punktów kontrolnych P_1, P_2 , oraz wyraża się wzorem:

$$b(t) = (1 - t)^3 * P_0 + 3t(1 - t)^2 * P_1 + 3t^2(1 - t) * P_2 + t^3 * P_3 \quad \text{dla } t \in [0, 1]$$

Aby zapewnić płynne połączenia pomiędzy segmentami, sąsiednie punkty kontrolne powinny znajdować się na tej samej prostej. Dla każdego punktu startowego wyznaczane są punkty kontrolne segmentu poprzednioego i następnego, na podstawie sąsiadujących punktów startowych.

Poniższy rysunek przedstawia trzy wylosowane punkty A, B, C , oraz proces generacji punktów kontrolnych dla środkowego punktu B . Na początku wyznaczana jest prosta prostopadła do dwusiecznej kąta pomiędzy wektorami BA i BC , poprzez normalizację ww. wektorów odpowiednio do BA' i BC' , a następnie odjęcie powstałych wektorów. Punkt kontrolny D znajduje się na ww. prostej, w odległości równej połowie długości wektora BA . Analogicznie położony jest punkt kontrolny E . Po wykonaniu powyższych operacji dla każdego punktu startowego na wyznaczonej trasie, w każdym segmencie znajdują się oba punkty kontrolne. Po podstawieniu punktów startowych i kontrolnych (np. A, F, D, B) dla każdego segmentu wyznaczona zostanie cała trasa.



Rysunek 1.1: Generowanie punktów kontrolnych krzywej Bezier

1.0.4 Stworzenie terenu 3D

Każdy stworzony teren gry jest rozmiaru 512×512 , z czego 10% przeznaczone jest na marginesy z każdej strony, co pozostawia wewnętrzną przestrzeń 409×409 na wygenerowany tor. Każdy punkt wygenerowanej pętli jest skalowany odpowiednio, aby trasa wypełniła całość dostępnej przestrzeni. Następnie dla każdego punktu terenu generowana jest wysokość, zdefiniowana jako procent maksymalnej wysokości 128. Wysokość składa się z trzech warstw, wykorzystujących Szum Perlina, każda coraz bardziej szczegółowa. Implementacja Szumu Perlina została dostarczona jako fragment biblioteki *Mathf* przez Unity [5].

Algorithm 1.3: Wyznaczenie wysokości terenu

Data:

```
x ← współrzędna X punktu;  
y ← współrzędna Y punktu;  
1 x ← x + offsetX;  
2 y ← y + offsetY;  
3 h1 ← detailsMain · PerlinNoise(x · scaleMain, y · scaleMain);  
4 h2 ← detailsMinor · PerlinNoise(x · scaleMinor, y · scaleMinor);  
5 h3 ← detailsTiny · PerlinNoise(x · scaleTiny, y · scaleTiny);  
6 return h1 + h2 + h3
```

Powyższa funkcja pozwala na sterowanie kształtem terenu poprzez parametry *details* i *scale* dla każdej z trzech warstw *main*, *minor*, *tiny*. Warstwa *main* definiuje ogólny kształt terenu, warstwa *minor* dodaje mniejsze pagórki, natomiast warstwa *tiny* dodaje drobne detale. Dodatkowo poprzez sterowanie *offsetX* i *offsetY* możliwe jest przesunięcie wszystkich wartości w płaszczyźnie poziomej.

1.0.5 Przypisanie tekstur

Zależnie od rodzaju terenu, wybierany jest odpowiedni zestaw tekstur. Dla każdego punktu wygenerowanego terenu przypisane są odpowiednie wartości przezroczystości tekstur, tworząc jak najbardziej realistyczne krajobrazy. Decyzja o wybraniu przezroczystości tekstur została podjęta na podstawie rodzaju terenu wybranego przez użytkownika, wysokości danego punktu, kierunku płaszczyzny zbocza, kąta nachylenia zbocza oraz odległości punktu od centrum wygenerowanej trasy. Odczytanie wartości takich

jak wysokość, kierunek płaszczyzny i kąt nachylenia można wykonać w stałym czasie $O(1)$. Natomiast znalezienie najbliższego punktu trasu jest bardziej złożonym procesem. Poniżej przedstawiono proces wyboru metody jego wyznaczania.

1.0.6 Znalezienie najbliższego punktu krzywej Bezier

Poniższa metoda dzieli każdy segment trasy na m odcinków, a następnie iteruje wszystkie końce odcinków we wszystkich segmentach. Optymalna ilość podziału wyznaczona została eksperymentalnie. Metoda ta ma złożoność czasową $O(nm)$, gdzie n oznacza liczbę segmentów trasy, natomiast m oznacza dokładność podziałów danego segmentu.

Algorithm 1.4: Znalezienie najbliższego punktu krzywej Bezier

```

Data:  $P \leftarrow \text{cel poza krzywą}$ 
1  $\minDist \leftarrow \text{infinity};$ 
2  $\minT \leftarrow 0;$ 
3 for  $i = 0; i \leq 100; i++ \text{ do}$ 
4    $t \leftarrow i \cdot 0.01;$ 
5    $pProjection \leftarrow B(t);$ 
6    $dist \leftarrow \text{distance}(P, pProjection);$ 
7   if  $dist < \minDist$  then
8      $\minDist \leftarrow dist;$ 
9      $\minT \leftarrow t;$ 
10 return  $B(\minT)$ 
```

1.0.7 Udoskonalone znalezienie najbliższego punktu krzywej Bezier

Poniższa metoda wywodzi się z artykułu [9], oraz została przystosowana dla krzywej Bezier 3 stopnia. Funkcja opisująca krzywą Bezier:

$$b(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

Pierwsza pochodna po t :

$$\frac{b}{dt}(t) = -3(1-t)^2 P_0 + 3(1-t)^2 P_1 - 6t(1-t)P_1 - 3t^2 P_2 + 6t(1-t)P_2 + 3t^2 P_3$$

Aby znaleźć rzut celu na krzywą, zależy znaleźć taki punkt na krzywej, aby styczna w tym punkcie była prostopadła do odcinka łączącego go z celem. Zatem dla zadanego punktu celu C i szukanej wartości t , określającej w którym miejscu krzywej znajduje się szukany rzut, zdefiniowana została funkcja f :

$$f(t) = (C - b(t)) \cdot \frac{b}{dt}(t)$$

Funkcja f wykorzystuje iloczyn skalarny, zatem szukane t zwraca wartość $f(t) = 0$. Ze względu na kształt krzywej, może na niej znajdować się kilka punktów będących rzutami prostopadłymi celu C , natomiast poszukiwany jest punkt najbliższej celu, zatem funkcję f należy przemożyć przez odległość od celu do znalezionej punktu.

$$f(t) = [(C - b(t)) \cdot \frac{b}{dt}(t)] \text{distance}(C, b(t))$$

Aby znaleźć najbliższy punkt na krzywej, należy znaleźć miejsce zerowe funkcji $f(t)$. W tym celu wykorzystano zmodyfikowaną metodę bisekcji. Tradycyjnie metoda bisekcji zakłada, że wartości funkcji na obu krańcach przedziału są przeciwnego znaku, następnie wybiera środkowy punkt i kontynuuje przeszukiwanie już w połowie przedziału. Natomiast w tym wypadku nie jest zagwarantowane, że funkcja będzie

miała przeciwnie znaki na końcach przedziału, dlatego gdy są tego samego znaku, liczona jest bisekcja dla obu podprzedziałów.

Algorithm 1.5: Udoskonalona bisekcja

Data:

```
a ← początek przedziału;  
b ← koniec przedziału;  
delta ← dokładność argumentów;  
epsilon ← dokładność wyników;  
maxit ← maksymalna ilość iteracji;  
1 if maxit < 1 then  
2   return (a+b)/2  
3 u ← f(a);  
4 v ← f(b);  
5 e ← (b - a)/2;  
6 c ← a + e;  
7 w ← f(c);  
8 if sing(u) != sign(v) then  
9   if abs(e) < delta or abs(w) < epsilon then  
10    return c  
11  if sing(u) != sign(w) then  
12    return UdoskonalonaBisekcja(a, c, delta, epsilon, maxit - 1)  
13  else  
14    return UdoskonalonaBisekcja(c, b, delta, epsilon, maxit - 1)  
15 else  
16  left ← UdoskonalonaBisekcja(a, c, delta, epsilon, maxit/2);  
17  right ← UdoskonalonaBisekcja(c, b, delta, epsilon, maxit/2);  
18  if abs(f(left)) < abs(f(right)) then  
19    return left  
20  else  
21    return right
```

1.0.8 Porównanie metod

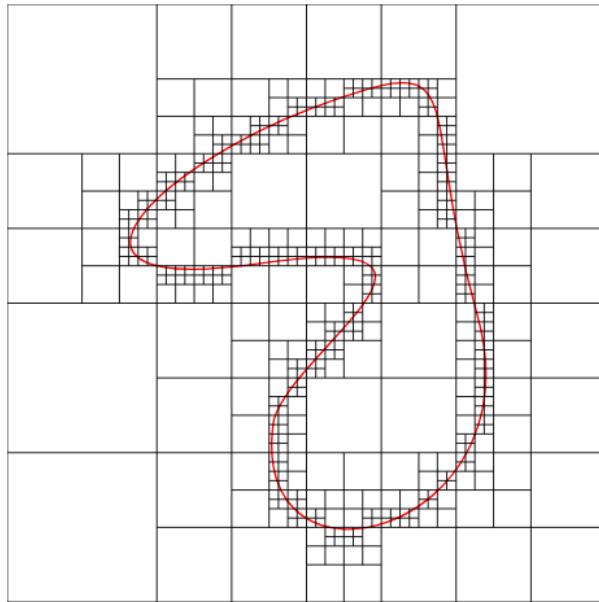
Metoda druga jest wykonywana szybciej, dodatkowo zapewniając większą szczegółowość.

Method 1		Method 2			lp
ilość podziałów	czas [ms]	ilość iteracji	odpowiadająca ilość podziałów	czas [ms]	
50	42046	5	$2^5 = 32$	30580	1
100	81421	10	$2^{10} = 1024$	59606	2
150	121224	15	$2^{15} = 32768$	70178	3

Tablica 1.1: Porównanie metod rzutowania punktu na krzywą Bezier

1.0.9 Optymalizacja czasu

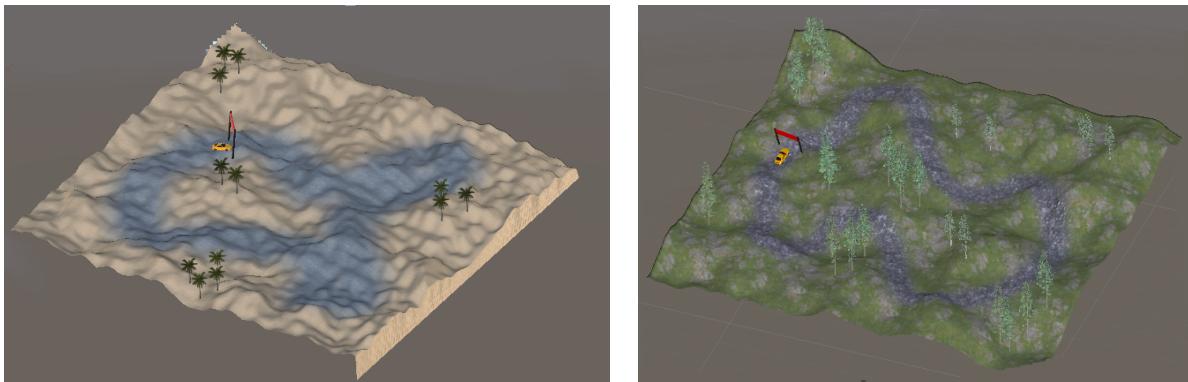
Zastosowanie wybranej metody dla każdego z 1048576 punktów (przy rozdzielczości terenu 1024×1024) trwa ponad godzinę i produkuje dużą liczbę nieużywanych informacji. Aby skrócić czas generowania terenu do minimum teren został podzielony na sekcje. Następnie, jeżeli środkowy punkt sekcji znajdował się w odległości od drogi mniejszej niż połowa przekątnej sekcji, tj. jeżeli droga przebiegała przez daną sekcję, była ona rekurencyjnie dzielona na podsekcje. Powyższe kroki powtórzono aż do otrzymania zadowalającej jakości.



Rysunek 1.2: Podział terenu w celu optymalizacji ilości obliczeń

1.0.10 Dodanie obiektów

Ostatnim etapem jest dodanie obiektów, takich jak meta oraz pojazdy. Domyślnie na każdym torze znajdują się cztery pojazdy, ustawione jeden za drugim, z czego trzy sterowane przez bota, a jeden przez gracza. Dodatkowo dodawane są krawędzie, których celem jest zapobieganie spadania pojazdu poza wygenerowany teren. W celu śledzenia przebiegu trasy, w równych odstępach dodawane są punkty kontrolne. Przykładowo otrzymano poniższy efekt:



Rysunek 1.3: Wygenerowane tereny

Rozdział 2

Wytrenowanie bota

2.0.1 Cel i funkcjonalności

Wytrenowany bot powinien być w stanie poruszać się po wygenerowanym wcześniej terenie, rozpoznając i trzymając się drogi. Trening bota powinien wykorzystywać metody uczenia przez wzmacnianie.

2.0.2 Wykorzystane technologie

Silnik Unity udostępnia dodatek open-source ML-Agents [8] [10], który pozwala na wykorzystanie środowisk utworzonych w grze do treningu modeli. Stanowi on warstwę łączącą Unity z biblioteką Tensorflow, oraz zapewnia wiele nowoczesnych algorytmów głębokiego uczenia przez wzmacnianie (m.in. Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), self-play). W powyższej pracy wykorzystano algorytm PPO, ze względu na większą stabilność treningu [11]. Śledzenie wielu metryk postępu uczenia było dostępnych z wykorzystaniem programu tensorboard.

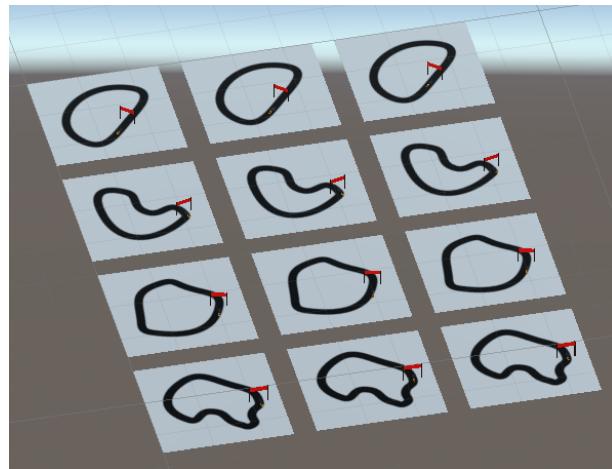
2.0.3 Uczenie przez wzmacnianie

Uczenie przez wzmacnianie jestem jednym z trzech głównych nurtów uczenia maszynowego, gdzie agent uczy się polityki optymalnej w danym środowisku metodą prób i błędów, otrzymując wyłącznie wartość nagrody jako informację zwrotną. W przeciwieństwie do uczenia nadzorowanego, metoda ta nie wymaga wcześniejszego przygotowania dużej ilości opisanych danych. Pozwala to na wprowadzenie bota do nieznanego środowiska i natychmiastowe podjęcie interakcji.

Cykl uczenia przez wzmacnianie opiera się na akcji bota i reakcji. Bot zbiera obserwacje na podstawie stanu w jakim się znajduje w środowisku. Następnie podejmuje decyzję na podstawie obserwacji. Po podjęciu decyzji wykonywana jest odpowiednia akcja, po czym za akcję przyznawana jest nagroda lub kara, w zależności czy akcja wpłynęła pozytywnie na przybliżanie się bota do celu. Na podstawie zbioru informacji zawierającego stan początkowy, podjęte akcje i otrzymaną nagrodę trenowana jest polityka, maksymalizując oczekiwany wynik.

2.0.4 Metodyka uczenia

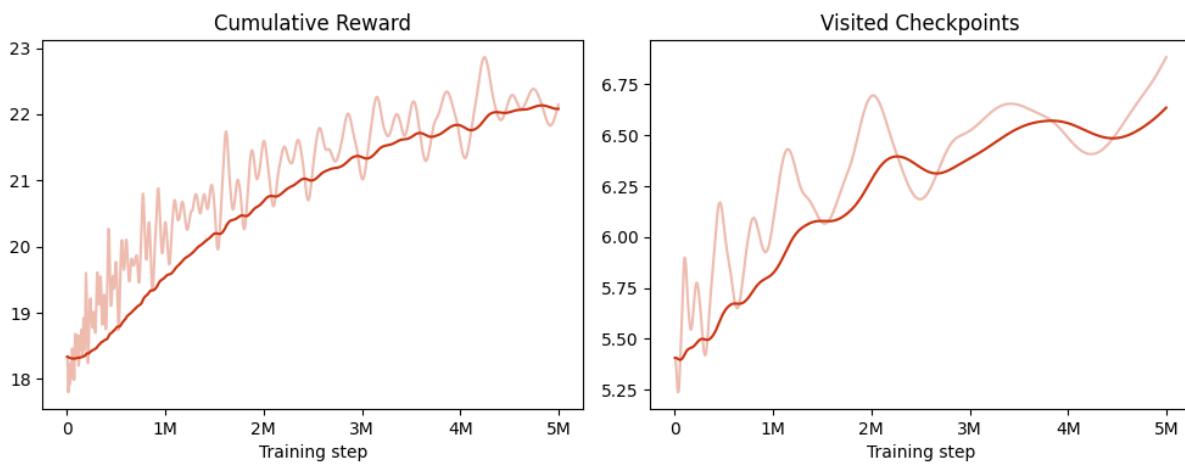
Wstępne uczenie odbywa się na terenach płaskich (wysokość każdego punktu jest równa zero) oraz w terenie ze zwiększym kontrastem - droga jest czarna a wszystko pozostałe jest białe. Do treningu uruchomione jest naraz 12 terenów, zawierających 4 różne trasy. Na każdym z terenów znajduje się pomiędzy 1 a 4 pojazdy, co oznacza że model jest trenowany na 12-48 niezależnych instancjach naraz, w celu przyspieszenia nauki. W kolejnych etapach zwiększany jest stopień trudności trasy, reprezentowany przez ilość i ostrość zakrętów. Następnie zwiększane jest pofałdowanie terenu, oraz zmieniane są tekstury drogi i terenu.



Rysunek 2.1: Prowadzenie wielu treningów na raz

2.0.5 Analiza wyników

Wyniki uczenia przedstawione zostały na wykresie za pomocą dwóch metryk. Wartość *Cumulative Reward* oznacza średnią wartość nagrody z jednego epizodu dla jednego agenta. Wartość *Visited punkt kontrolny* informuje jak dużą część trasy przeszedł agent, uśrednioną z jednego epizodu dla jednego agenta. Pełne okrążenie zawsze w sobie 40 punktów kontrolnych. Wartości zaznaczone na wykresach bledszym kolorem są wartościami rzeczywistymi, natomiast wartości pogrubione wyliczone zostały z wykorzystaniem średniej kroczącej eksponentjalnej, aby lepiej uwidoczyć tendencje zmian.



Rysunek 2.2: Przykładowa analiza wyników

2.0.6 Wybór obserwacji

Wybór informacji, jakie dostępne są dla bota znacznie wpływa na podejmowane przez niego decyzje. Za mała ilość informacji nie pozwoli na wykrycie odpowiedniej ilości szczegółów trasy, natomiast za duża znacznie zwiększy czas treningu i inferencji, wprowadzając niepotrzebny szum.

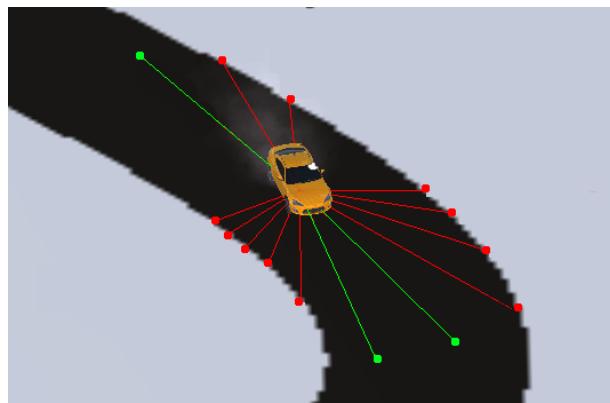
Dane wynikające z trasy

Pierwszy sposób obserwacji opiera się na wartościach, do których zwykły gracz nie ma dostępu, ponieważ wymagają m.in. znajomości dokładnego wzoru krzywej trasy do wyliczenia. Dane, które są przekazywane do modelu jako obserwacje to:

- odległość w linii prostej od aktualnej pozycji do centrum szerokości trasy
- kąt pomiędzy kierunkiem jazdy pojazdu a kierunkiem stycznej do toru
- odległość w linii prostej od aktualnej pozycji do najbliższego punktu kontrolnego
- kąt pomiędzy kierunkiem jazdy pojazdu a kierunkiem do najbliższego punktu kontrolnego
- kąt nachylenia terenu
- aktualna pozycja kół (-1 oznacza maksymalnie skręcone w lewo, 1 maksymalnie w prawo)
- aktualny stopień wciśnięcia pedału gazu (-1 oznacza jazdę do tyłu, 1 jazdę do przodu)
- aktualna prędkość

Dane odległości od krawędzi

Podobnie jak większość samochodów autonomicznych, poniższa metoda wykorzystuje wizualizację przestrzeni na podstawie odległości, nazywaną lidar [4]. Technologia ta w realnym świecie buduje model otoczenia wysyłając wiązki laserowe i mierząc trasę przebytą przez nie do przeszkody. Analogicznie, symulowany pojazd testuje odległości od krawędzi trasy oraz ewentualnych przeszkód, takich jak inne pojazdy.

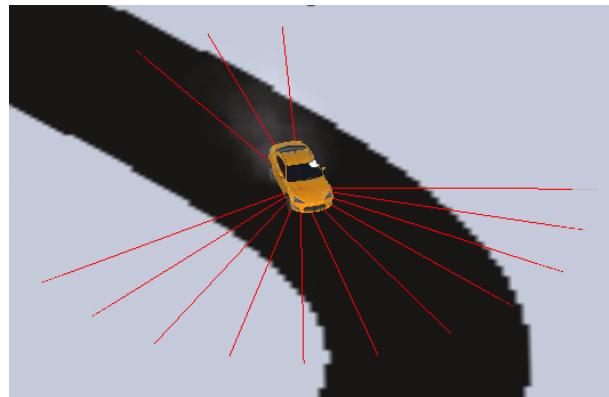


Rysunek 2.3: Obserwacje bota dystansów naokoło pojazdu

Dodatkowo przekazywana jest aktualna prędkość pojazdu jako obserwacja, zważywszy że nie da się jej w żaden sposób odczytać z jednej klatki pomiarów.

Dane wizualne jednowymiarowe

Poniższe podejście testuje metodę obserwacji wykorzystującą bodźce wizualne. W tym przypadku tworzona jest lista wartości kolorów dla każdego punktu w stałej odległości od pojazdu. Generuje to okrąg kolorów naokoło pojazdu, który powinien być w stanie przekazać takie informacje jak zakręty na trasie, będąc znacznie mniejszego rozmiaru niż obraz z kamery. Na rysunku zaznaczono kilka promieni pobierających kolory otoczenia.

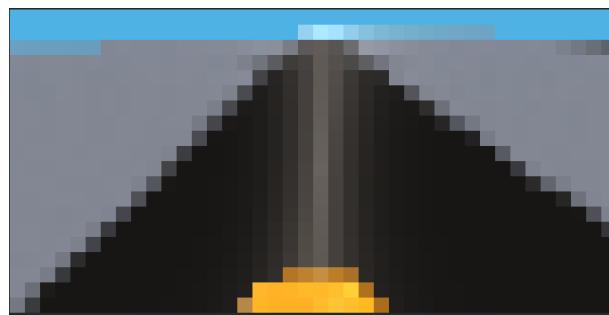


Rysunek 2.4: Obserwacje bota kolorów naokoło pojazdu

Dodatkowo przekazywana jest aktualna prędkość pojazdu jako obserwacja, zważywszy że nie da się jej w żaden sposób odczytać z danych wizualnych.

Dane wizualne z kamery przedniej

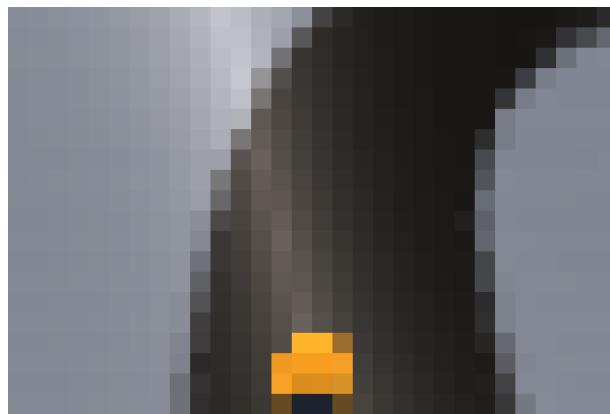
Wykorzystując imformacje w postaci w jakiej są one widoczne dla każdego gracza, kolejna metoda pobiera obraz z kamery jako obserwację. Kamera umiejscowiona jest na przedzie pojazdu, zapewniając perspektywę podobną do zwykłego prowadzenia auta. Aby ograniczyć ilość informacji dostarczanych do sieci, obraz z kamery skalowany jest do rozdzielczości 40×20 oraz konwertowany na skalę szarości. Dodatkowo, jak powyżej, przekazywana jest aktualna prędkość pojazdu.



Rysunek 2.5: Obserwacje bota z kamery przedniej

Dane wizualne z kamery z lotu ptaka

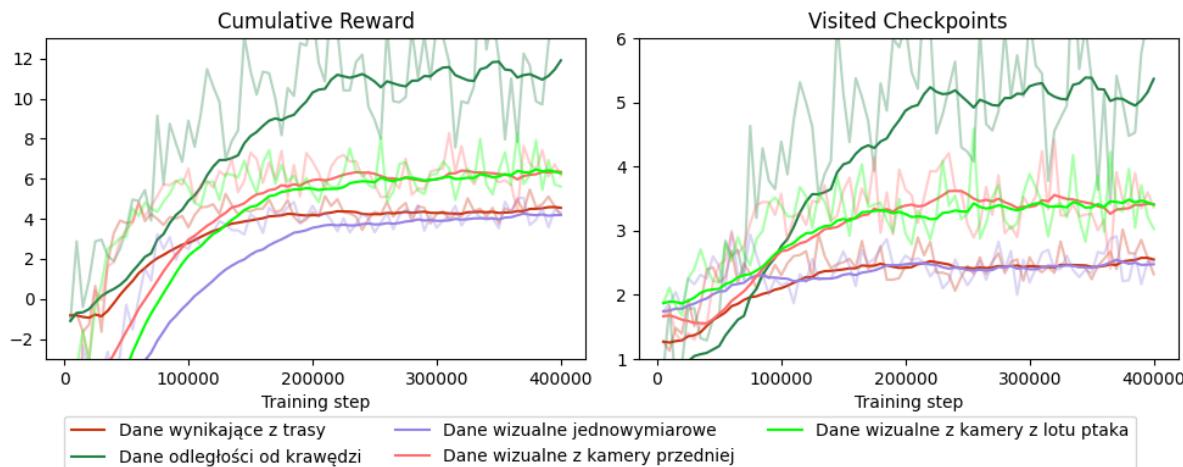
Wykorzystując imformacje widoczne przez kamerę z lotu ptaka model powinien być w stanie łatwiej zauważać krzywiznę toru. Tak jak powyżej, obraz z kamery skalowany jest do rozdzielczości 30×20 oraz konwertowany na skalę szarości, oraz przekazywana jest aktualna prędkość pojazdu.



Rysunek 2.6: Obserwacje bota z kamery z lotu ptaka

Porównanie

Poniżej znajdują się wykresy porównujące szybkość uczenia się w pierwszych 400 000 iteracjach. Wykres pierwszy pokazuje otrzymaną nagrodę za akcję, uśrednioną dla każdego agenta, dla każdej akcji. Drugi wykres przedstawia jak daleko udało się dojechać. Na przestrzeni całego toru rozłożone jest 40 punktów kontrolnych. Celem jest maksymalizacja obu wartości.



Rysunek 2.7: Porównanie metod obserwacji

Najlepsze wyniki otrzymano dla metody wykorzystującej odległość od przeszkód. Metoda ta zbiera łącznie 14 dystansów, co wystarcza do zawarcia najpotrzebniejszych informacji. Dane z kamery okazały się zbyt obszerne (odpowiednio 600 i 800 pikseli), spowalniając trening.

2.0.7 Wybór akcji

Po podjęciu decyzji bot musi wykonać odpowiednią akcję. Bot podejmuje decyzję w dwóch płaszczyznach. Pierwsza odnosi się do przemieszczania przód-tył, natomiast droga odpowiada skręceniu kierownicy prawo-lewo. Decyzje te mogłyby być wybierane z przestrzeni ciągłych lub dyskretnych.

Akcje w przestrzeni ciągłej

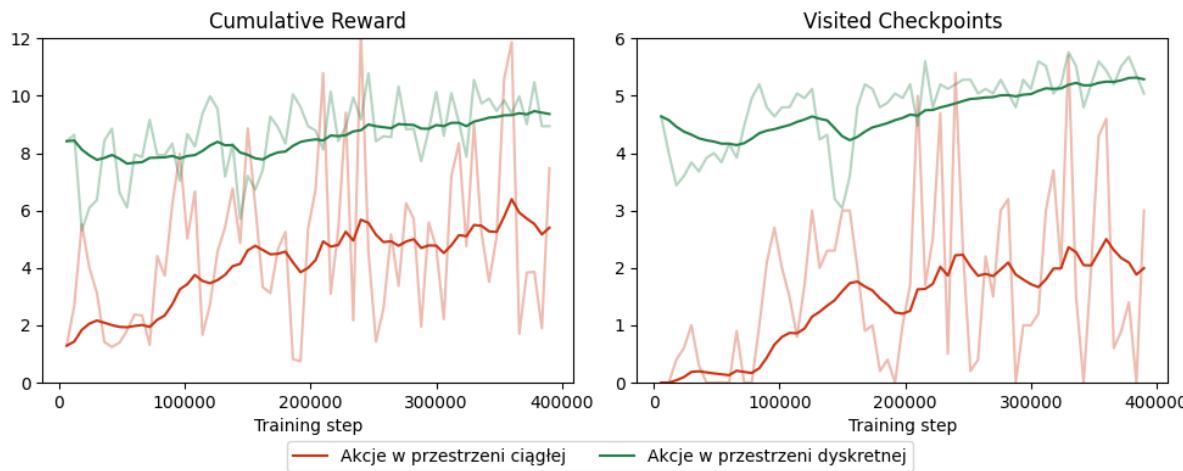
Dla każdej płaszczyzny bot zwraca wartość rzeczywistą z zakresu $[-1, 1]$. W pierwszej płaszczyźnie wartość odpowiada dokładnie stopniowi wciśnięcia pedału gazu, natomiast w drugiej - kątowi skrętu kierownicy. Poniżej bot jest w stanie zawsze ustawić konkretną wartość, przejście pomiędzy kolejnymi akcjami nie muszą być płynne - w pierwszej klatce bot może skierować kierownicę 30 stopni w lewo, natomiast już klatkę później może ustawić ją na 20 stopni w prawo, bez stanów przejściowych.

Akcje w przestrzeni dyskretnej

Dla każdej płaszczyzny bot zwraca wartość całkowitą ze zbioru $-1, 0, 1$. W pierwszej płaszczyźnie wartość -1 odpowiada zwiększeniu nacisku na pedał hamulca, 0 oznacza brak zmian, 1 oznacza zwiększenie nacisku na pedał gazu. Analogicznie w drugiej płaszczyźnie, -1 odpowiada skręceniu kierownicy mnożonej w lewo, 1 mocniej w prawo, a 0 brak zmian. W ten sposób zmiany są bardziej płynne w czasie.

Porównanie

Zastosowanie akcji dyskretnych pozwoliło botowi na znacznie łatwiejsze poruszanie się w środowisku, co przełożyło się na znacznie lepsze wyniki w pierwszych 400 000 krokach treningu.



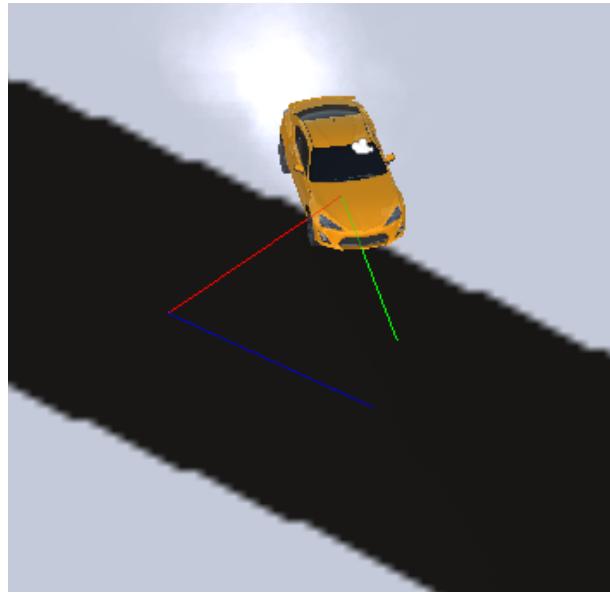
Rysunek 2.8: Porównanie metod podejmowania akcji

2.0.8 Zdefiniowanie funkcji oceny

Ocena powinna odzwierciedlać w jakim stopniu wykonana akcja przybliżyła bota do osiągnięcia celu. W każdym cyklu, podjęta akcja oceniana jest na podstawie trzech parametrów.

$$\begin{aligned} Q = & (0.5 - \text{distanceToRoadCenter}) \cdot 0.1 + \\ & (0.05 - \text{abs}(\text{angleToTangent})) + \\ & (\text{distanceTraveledInFrame} - 0.1) + \\ & 0.01 \end{aligned}$$

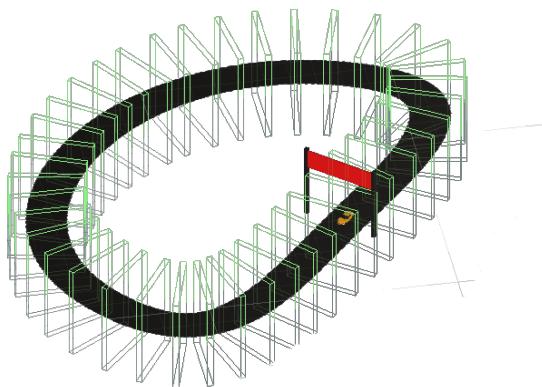
Po pierwsze przyznawana jest nagroda za utrzymywanie się w odległości mniejszej niż 50% szerokości drogi od środka drogi (linia czerwona na poniższym rysunku), w przeciwnym przypadku kara, wprost proporcjonalna do ww. odległości. Po drugie przyznawana jest nagroda za utrzymywanie kierunku jazdy (linia zielona na poniższym rysunku) w zakresie ± 10 stopni od kierunku trasy (linia niebieska na poniższym rysunku), w przeciwnym przypadku kara, wprost proporcjonalna do w.w kąta. Po trzecie przyznawana jest nagroda za dystans pokonany od ostatniej akcji, jeżeli wynosi co najmniej 0.1. Dodatkowo przyznawana jest nagroda za każdą klatkę, aby promować jak najdłuższe treningi.



Rysunek 2.9: Wizualizacja oceny akcji

Dodatkowo bot oceniany jest za każdym razem kiedy wejdzie w interakcję z innym obiektem. Na około całego terenu rozmieszczone są bariery, które powodują koniec epizodu po dotknięciu i powrót pojazdu na pozycję początkową, dodatkowo dodając karę 0.1. Na pierwszym etapie treningu takie same bariery ustawione są wzduł drogi, tak że pojazd kończy epizod za każdym razem kiedy zjedzie z drogi. Po wytrenowaniu bota w takich warunkach bariery te zostaną ściągnięte, aby zobaczyć czy mimo to bot będzie trzymał się drogi.

Wzdłuż całej trasy rozstawione są punkty kontrolne, które pojazd powinien przebyć w odpowiedniej kolejności. Za każdy punkt kontrolny zaliczony w kolejności nagroda zwiększa się o 1. Jeżeli pojazd zahaczy o któryś z 4 sąsiednich punktów kontrolnych (2 w tył, 2 w przód) dostaje tylko karę 0.1, natomiast jeżeli dotknie innego punktu kontrolnego, epizod jest kończony z karą -1.



Rysunek 2.10: Rozmieszczenie punktów kontrolnych

2.0.9 Dobranie parametrów

Wybrany algorytm Proximal Policy Optimization pozwala na osiągnięcie nie najgorszych wyników, natomiast dostosowanie hyperparametrów potrafi znacznie poprawić efekty treningu.

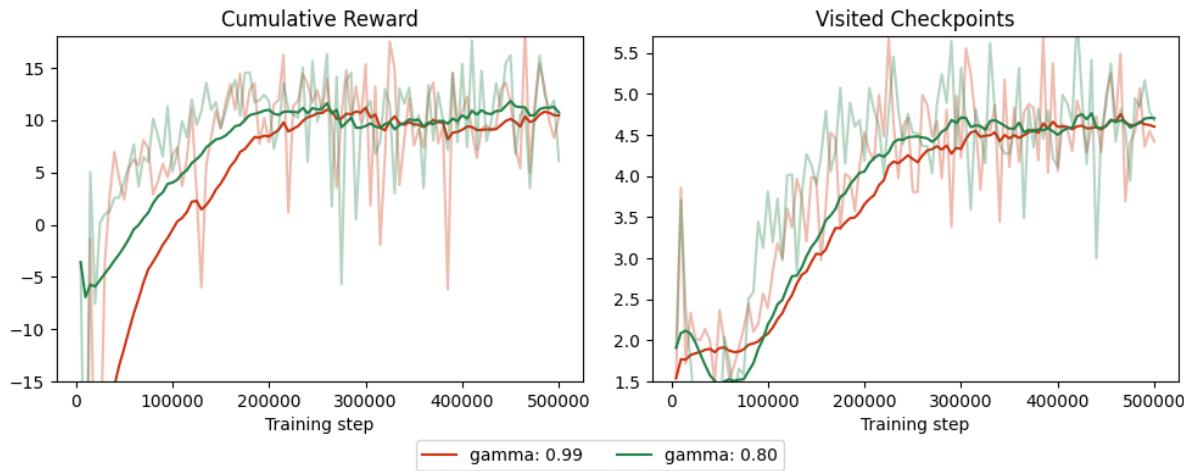
Algorithm 2.1: Hiperparametry początkowe treningu

```
1 hyperparameters :  
2     batch_size : 1024  
3     buffer_size : 10240  
4     learning_rate : 0.0003  
5     beta : 0.005  
6     epsilon : 0.2  
7     lambda : 0.95  
8     num_epoch : 3  
9     learning_rate_schedule : linear  
10 reward_signals :  
11     extrinsic :  
12         gamma : 0.99  
13         strength : 1.0
```

Poniżej zostały przedstawione wybory poszczególnych parametrów. Porównane zostały konfiguracje dla pierwszych 500000 iteracji, za każdym razem zaczynając od losowych wag. Po wyborze optymalnej wartości parametru, była ona aktualizowana i wykorzystywana w kolejnych testach jako domyślna.

2.0.10 gamma

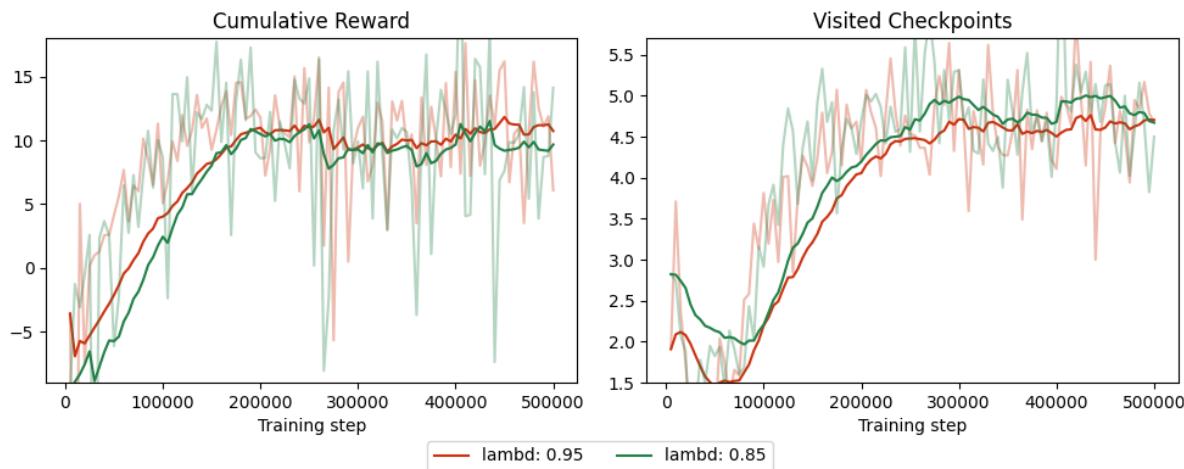
Parametr γ można rozumieć jako jak daleko w przyszłość agent powinien dbać o możliwe nagrody. W sytuacjach gdy agent powinien podejmować decyzje w oczekiwaniu na przewidywane nagrody w przyszłości, wartość γ powinna być większa (0.995), natomiast jeżeli agent powinien bardziej dbać o nagrody natychmiastowe wartość γ może osiągać niższe wartości (0.8). Po porównaniu granic typowego zakresu wartości, na wykresach poniżej widać, że wartość niższa przekłada się na minimalnie lepszy trening.



Rysunek 2.11: Wybór hiperparametrów: gamma

2.0.11 lambda

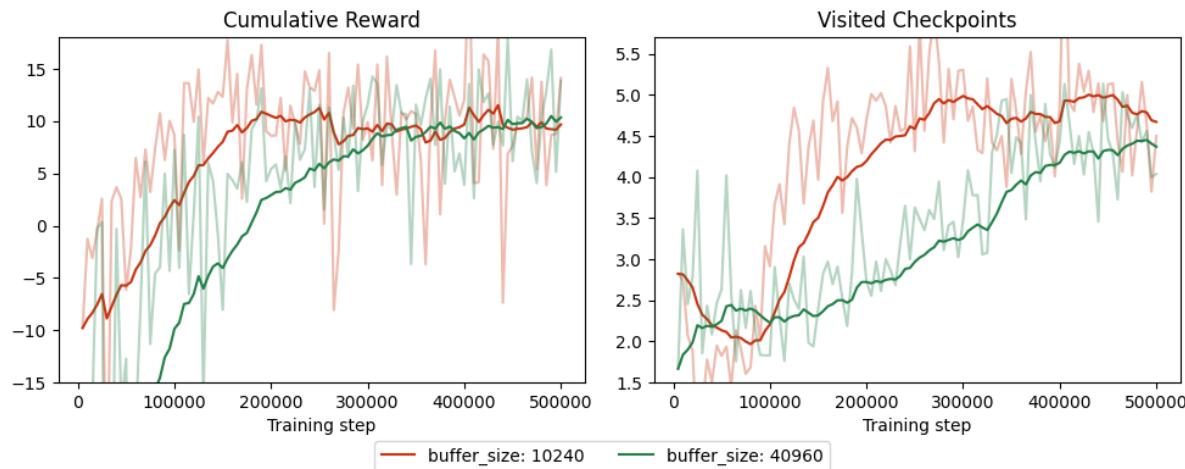
Parametr λ definiuje w jakim stopniu agent polega na przewidzianej wartości podczas przewidywania kolejnej wartości. Niższe wartości λ oznaczają poleganie w większym stopniu na przewidzianej wartości, natomiast większa λ odpowiada poleganiu bardziej na rzeczywiście otrzymanych nagrodach. Po porównaniu granic typowego zakresu wartości (0.85 – 0.95), na wykresach poniżej widać, że wartość niższa przekłada się na minimalnie lepszy trening.



Rysunek 2.12: Wybór hiperparametrów: lambd

2.0.12 buffer size

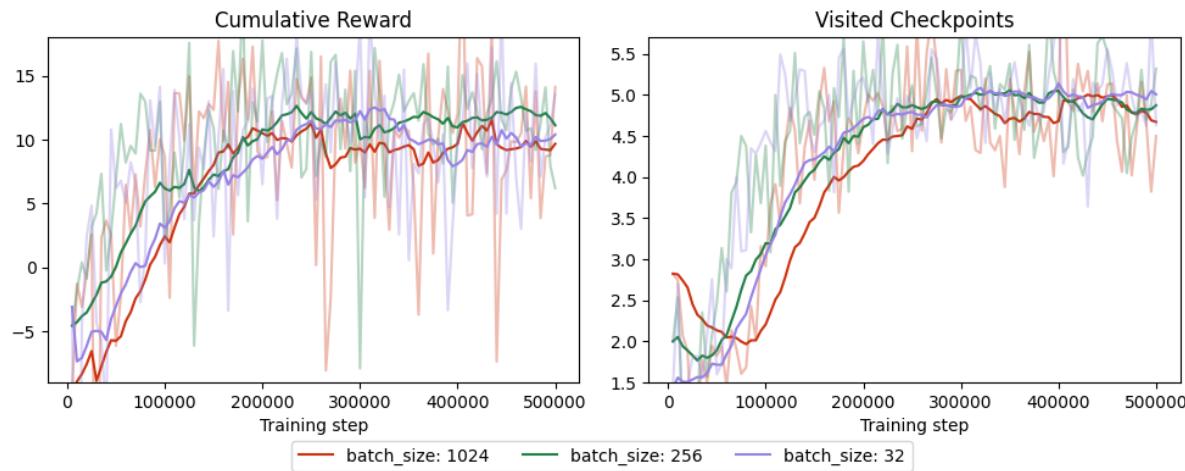
Rozmiar bufora odpowiada ilości cykli treningowych (zebranie obserwacji, podjęcie akcji, otrzymanie nagrody) które są zbierane przed aktualizacją modelu. Większa wartość przekłada się na bardziej stabilny trening, kosztem czasu. Po porównaniu wartości 10240 i 40960 można zauważać, że niższa wartość zapewnia zadowalającą stabilność, otrzymując podobne wartości znacznie wcześniej.



Rysunek 2.13: Wybór hiperparametrów: buffer size

2.0.13 batch size

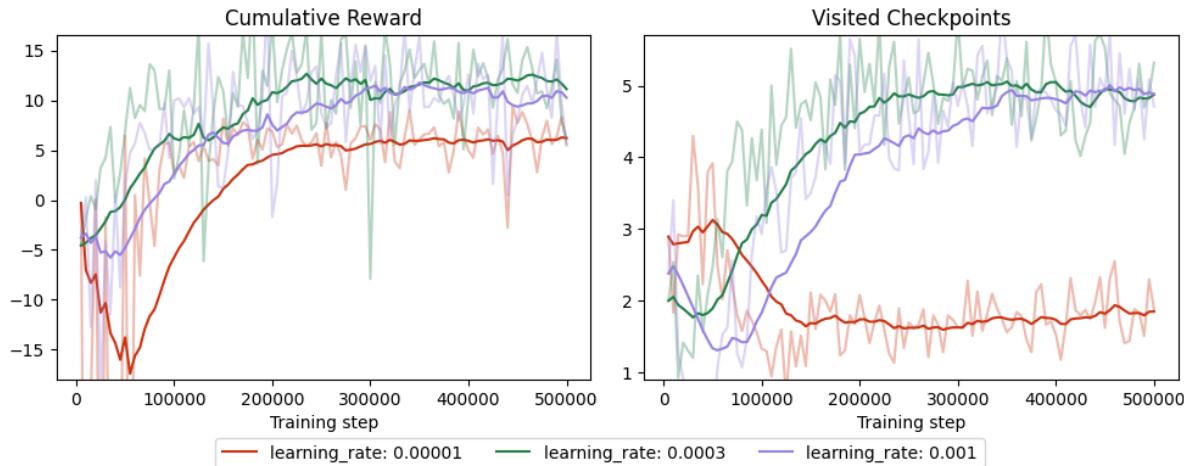
Parametr *batch_size* definiuje ilość cykli treningowych wykorzystywanych przy propagacji wstecznej. Przy dyskretnej przestrzeni akcji wartość ta powinna być mniejsza, niż dla akcji ciągłych. Zmniejszenie *batch_size* do 256 minimalnie poprawiło trening, natomiast spadek do 32 nie wprowadził zauważalnych różnic.



Rysunek 2.14: Wybór hiperparametrów: batch size

2.0.14 learning rate

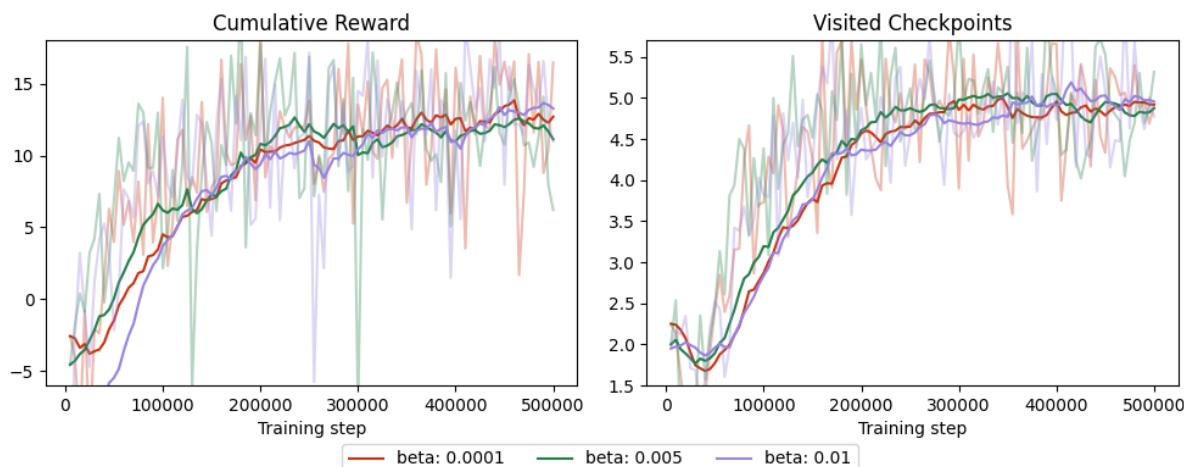
Prędkość uczenia bezpośrednio odnosi się do siły każdego kroku propagacji wstecznej. Analizując poniższe wykresy, wartość pośrodku typowego zakresu ($1e - 3, 1e - 5$) pozwala na najbardziej optymalną prędkość uczenia w skali 500000 iteracji.



Rysunek 2.15: Wybór hiperparametrów: learning rate

2.0.15 beta

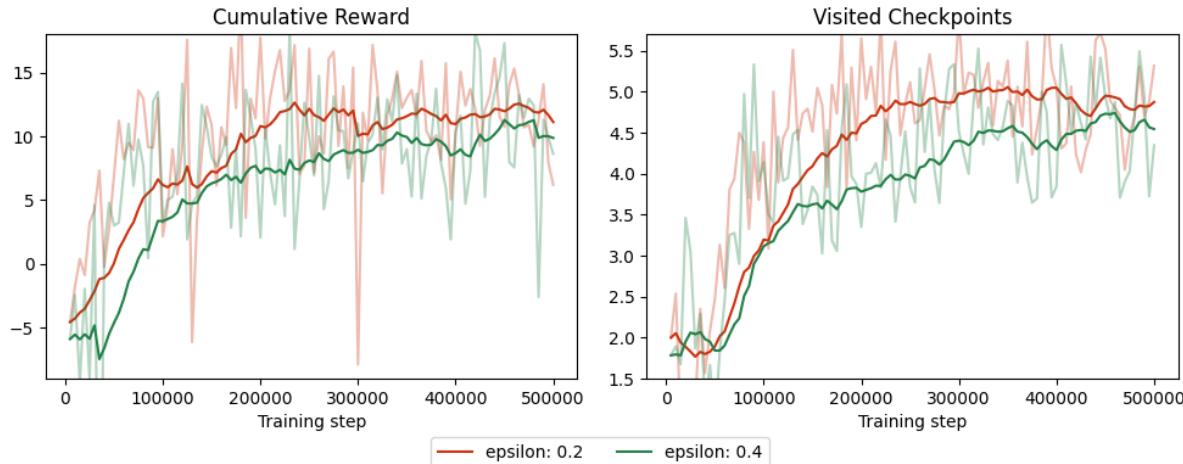
Parametr *beta* definiuje skalę randomizacji polityki. Większe wartości parametru powodują podejmowanie przez bota większej ilości losowych akcji, zwiększając ilość sprawdzanych rozwiązań. Dla zadanego problemu niższa *beta* powoduje szybsze unormowanie się wartości nagrody oraz postępu trasy, dlatego została wybrana wartość 0.005.



Rysunek 2.16: Wybór hiperparametrów: beta

2.0.16 epsilon

Zmienna *epsilon* informuje w jakim stopniu akceptowane są rozbieżności pomiędzy starą i nową polityką podczas propagacji wstecznej. Niższe wartości powodują bardziej stabilny postęp kosztem czasu treningu. Zwiększenie wartości do 0.4 nie spowodowało jednak przyspieszenia treningu.

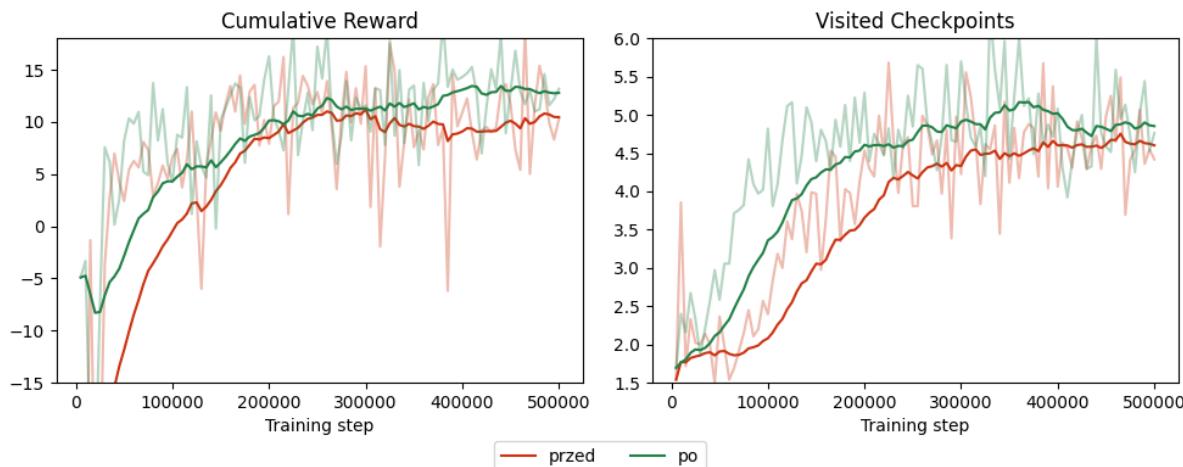


Rysunek 2.17: Wybór hiperparametrów: epsilon

Finalnie zostały wybrane poniższe hiperparametry, co pozwoliło na zauważalną poprawę treningu.

	Funkcja nagrody			Postęp trasy		
	przed	po	zmiana	przed	po	zmiana
średnia z wszystkich iteracji	7.38	9.79	+32.5%	3.91	4.49	+14.8%
średnia z najlepszych 99%	7.81	10.15	+29.9%	3.93	4.52	+15.0%
średnia z najlepszych 90%	9.63	11.49	+19.3%	4.15	4.73	+13.9%
średnia z najlepszych 50%	12.38	13.99	+13.0%	4.75	5.13	+8.0%
maksimum	19.04	19.64	+3.1%	6.18	6.35	+2.7%

Tablica 2.1: Wybór hiperparametrów: porównanie wyników



Rysunek 2.18: Wybór hiperparametrów: porównanie wyników

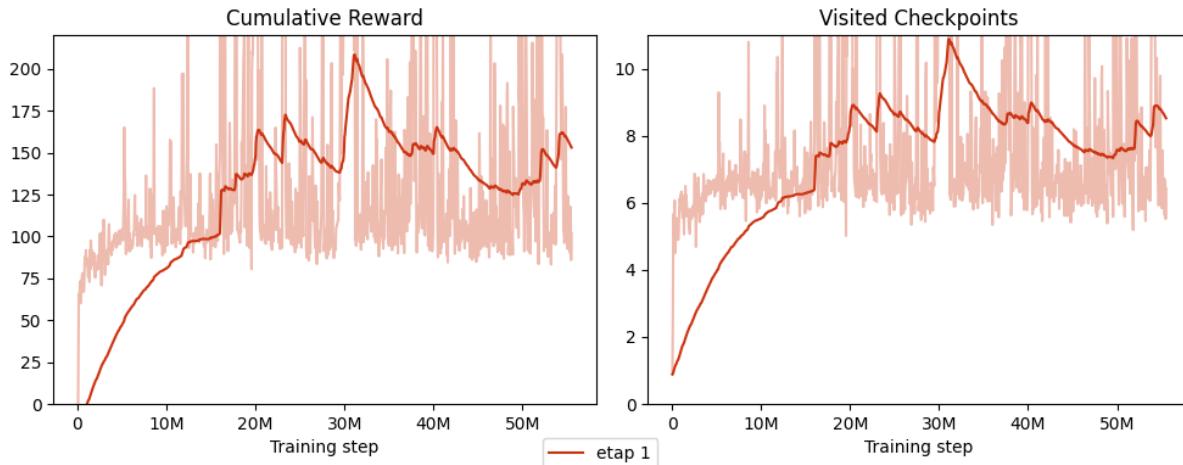


Algorithm 2.2: Hiperparametry finalne treningu

```
1 hyperparameters :  
2     batch_size : 256  
3     buffer_size : 10240  
4     learning_rate : 0.0003  
5     beta : 0.005  
6     epsilon : 0.2  
7     lambd : 0.85  
8     num_epoch : 8  
9     learning_rate_schedule : linear  
10 reward_signals :  
11     extrinsic :  
12         gamma : 0.8  
13         strength : 1.0
```

2.0.17 Dalsze etapy treningu

Przy trenowaniu ważne jest, aby początkowe zadania były proste, aby agent był w stanie je łatwo wykonać. Następnie należy stopniowo zwiększać poziom trudności, dając agentowi odpowiednio dużo czasu na przystosowanie się do zmian. Dlatego pierwszy etap treningu polega na przejechaniu trasy będącej okręgiem, po płaskim terenie, bez wykorzystania tekstur (droga jest czarna, otoczenie białe) oraz bez przeciwników (w każdym środowisku znajduje się jeden agent).



Rysunek 2.19: Postęp treningu: etap 1

Etap 2 zakłada dodanie większej różnorodności torów, zachowując stosunkowo prosty kształt, tylko nieznacznie odbiegający od okręgu, z małą ilością ostrych zakrętów.

Etap 3 sprawdza adaptowalność pojazdu na skomplikowanym torze, z dużą ilością ostrych zakrętów.

Etap 4 zakłada dodanie nierówności terenu, tworząc góry i pagórki na trasie.

Podsumowanie

W powyższej pracy została stworzona gra wyścigowa, wraz z botem. Bot został wytrenowany z wykorzystaniem uczenia przez wzmacnianie przy pomocy Unity ML-Agents. Jako interakcję ze środowiskiem bot posiada do dyspozycji dwie akcje w przestrzeni dyskretnej o wartościach -1, 0, 1 definiujące poruszanie się do przodu/tyłu oraz kąt skrętu kierownicy. Pojazd porusza się zgodnie z uproszczonymi zasadami fizyki symulowanymi przez silnik Unity. Jako obserwacje przyjmuje odległości pojazdu od ewentualnych przeszkód oraz krawędzi drogi i aktualną prędkość. Taki zbiór danych wejściowych ma na celu symulować podejście wykorzystywane aktualnie w samochodach autonomicznych.

Proces implementacji bota, wraz z wyborem obserwacji, podejmowanych akcji, funkcji nagrody i wyboru hiperparametrów przedstawiony został na wykresach ilustrujących efekty podjętej decyzji. W pracy zostały również opisane kolejne etapy treningu bota.



Bibliografia

- [1] Instalacja pakietu ml-agents. URL: <https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Installation.md>.
- [2] Instalacja silnika unity. URL: <https://unity.com/download>.
- [3] Kod Źródłowy gry. URL: <https://github.com/KamilMatejuk/Praca-inzynierska-2/>.
- [4] Lidar. URL: <https://pl.wikipedia.org/wiki/Lidar>.
- [5] Perlin noise. URL: <https://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html>.
- [6] Poziomy autonomii pojazdów sae. URL: <https://www.sae.org/blog/sae-j3016-update>.
- [7] Unity asset store. URL: <https://assetstore.unity.com/>.
- [8] Unity ml-agents repository. URL: <https://github.com/Unity-Technologies/ml-agents/>.
- [9] Xiao-Diao Chen et al. Improved algebraic algorithm on point projection for bézier curves. *Proceedings of the Second International Multi-Symposiums on Computer and Computational Sciences*, pages 158–163, 2007. URL: <https://hal.inria.fr/file/index/docid/518379/filename/Xiao-DiaoChen2007c.pdf>, doi:[10.1109/IMSCCS.2007.17](https://doi.org/10.1109/IMSCCS.2007.17).
- [10] Arthur Juliani et al. Unity: A general platform for intelligent agents. 2020. URL: <https://arxiv.org/pdf/1809.02627v2.pdf>.
- [11] Thomas Nakken Larsen et al. Comparing deep reinforcement learning algorithms' ability to safely navigate challenging waters. *FrontRobotAI*, 2021. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8473616/>, doi:[10.3389/frobt.2021.738113](https://doi.org/10.3389/frobt.2021.738113).



Załącznik A

Uruchomienie gry

Kod źródłowy programu, wraz z plikami wykonywalnymi gry, znajduje się na załączonej płycie CD, oraz online w repozytorium GitHub [3].

A.1 Struktura plików

Struktura plików wygląda następująco:

```
1 ./
2 |— Kod_Źródłowy/
3 |   |— Assets/
4 |
5 |— Gra/
6 |   |— Windows/
7 |   |— Linux/
8 |
9 |— Praca_Inżynierska.pdf
10 |— ...
```

A.2 Uruchomienie gry

Wewnątrz folderu *Gra* znajdują się pliki wykonywalne, pozwalające na uruchomienie gry.

Windows

Gra została przetestowana na systemie Windows 10. Aby uruchomić grę należy uruchomić plik *Gra/Windows/gra.exe*.

Linux

Gra została przetestowana na systemie Ubuntu 20.04. Aby uruchomić grę należy uruchomić plik *Gra/Linux/gra.x86_64*.

A.3 Kod źródłowy

Program został stworzony z wykorzystaniem silnika Unity, który sam dodaje znaczną ilość kodu, dla tego w katalogu *Kod_Źródłowy* został załączony jedynie folder zawierający autorską część kodu.

Wewnątrz znajduje się również folder *Assets/Other_Assets* zawierający zainportowane modele 3D ze sklepu Unity [7].

Aby uruchomić grę z kodu źródłowego, należy:

- zainstalować program UnityHub zgodnie z instrukcjami z oficjalnej strony [2].
- zainstalować pakiet MI-Agents [1].
- uruchomić Unity i stworzyć nowy projekt.
- podmienić folder *Assets* w nowym projekcie na pobrany folder *Kod_Źródłowy/Assets/*.
- wybrać opcję *MyTools > Custom Linux Build* lub *MyTools > Custom Windows Build*
- załadować scenę *Assets/Scenes/MainMenu*, oraz uruchomić.

A.4 Poruszanie się w grze

Po uruchomieniu gry użytkownikowi prezentowany jest ekran główny, wraz z dwiema opcjami. Pierwsza z nich pozwala na rozpoczęcie gry, natomiast druga - na utworzenie własnego terenu do gry.



Rysunek A.1: Instrukcja grania - ekran główny

Jeżeli użytkownik wybierze opcję gry, na kolejnym etapie zostanie poproszony o wybór poziomu oraz terenu. Tereny utworzone przez użytkownika też pojawią się poniższym menu rozwijanym.



Rysunek A.2: Instrukcja grania - wybór poziomu

Po włączeniu gry, wyścig rozpoczyna się od razu po dotknięciu ziemi. Do kontroli pojazdu należy używać strzałek na klawiaturze.



Rysunek A.3: Instrukcja grania - gra

Aby zatrzymać wyścig w trakcie, należy wcisnąć przycisk *Escape* na klawiaturze. Czas w grze zostanie zatrzymany aż do wyłączenia widocznego menu. Można to osiągnąć wciskając ponownie *Escape*, lub wybierając opcję *Resume* w menu. Dodatkowo użytkownik ma możliwość zakończyć grę i powrócić do głównego menu.



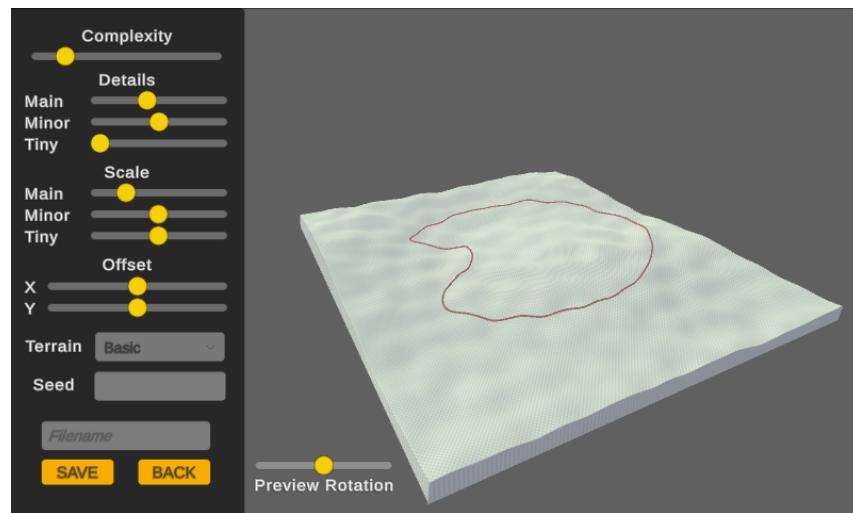
Rysunek A.4: Instrukcja grania - zatrzymanie gry

Oprócz domyślnie zdefiniowanych poziomów i terenów, użytkownik ma możliwość wygenerować własną trasę, za pomocą kilku parametrów:

- Parametry *Complexity* i *Seed* definiują jak bardzo kręta będzie droga.
- Parametry *Details*, *Scale* i *Offset* pozwalają na określenie kształtu terenu.
- Parametr *Terrain* określa szatę graficzną terenu.

Aby podejrzeć trasę z różnych stron można wykorzystać suwak *PreviewRotation*.

Po ustawieniu wybranych parametrów, należy ustawić nazwę pliku w polu *Filename*. Pod taką nazwą pojawi się wygenerowany teren podczas wyboru poziomu przed grą. Następnie należy kliknąć przycisk *Save* aby zapisać teren.



Rysunek A.5: Instrukcja grania - tworzenie poziomu



Załącznik B

Dokumentacja techniczna wygenerowana przez Doxygen (ENG)

B.1 Namespace Index

Here is a list of all documented namespaces with brief descriptions:

RacingGameBot	??
RacingGameBot.Data	36
RacingGameBot.Editors	36
RacingGameBot.Menu	36
RacingGameBot.Play	36
RacingGameBot.Terrains	37
RacingGameBot.Tests	37
RacingGameBot.Utils	37



B.2 Hierarchical Index

This inheritance list is sorted roughly, but not completely, alphabetically:

RacingGameBot.Terrains.Loop	51
RacingGameBot.Terrains.OrientedPoint	54
Agent	
RacingGameBot.Play.CarAgent	49
Editor	
RacingGameBot.Editors.TerrainGeneratorEditor	40
RacingGameBot.Editors.TerrainLoaderEditor	40
RacingGameBot.Editors.UpdatableDataEditor	41
IPointerClickHandler	
RacingGameBot.Menu.ButtonActions	41
IPointerEnterHandler	
RacingGameBot.Menu.ButtonActions	41
IPointerExitHandler	
RacingGameBot.Menu.ButtonActions	41
MonoBehaviour	
RacingGameBot.Menu.ButtonActions	41
RacingGameBot.Menu.CreateLevelMenu	42
RacingGameBot.Menu.EndMenu	45
RacingGameBot.Menu.MainMenu	46
RacingGameBot.Menu.PauseMenu	48
RacingGameBot.Play.MultiTraining	51
RacingGameBot.Terrains.TerrainGenerator	55
RacingGameBot.Terrains.TerrainLoader	58
RacingGameBot.Tests.BezierTests	60
RacingGameBot.Tests.TextureTests	61
RacingGameBot.Utils.Objects	65
ScriptableObject	
RacingGameBot.Data.UpdatableData	39
RacingGameBot.Data.TerrainGenData	38



B.3 Class Index

Here are the classes, structs, unions and interfaces with brief descriptions:

RacingGameBot.Data.TerrainGenData	38
RacingGameBot.Data.UpdatableData	39
RacingGameBot.Editors.BuildScript	39
RacingGameBot.Editors.TerrainGeneratorEditor	40
RacingGameBot.Editors.TerrainLoaderEditor	40
RacingGameBot.Editors.UpdatableDataEditor	41
RacingGameBot.Menu.ButtonActions	41
RacingGameBot.Menu.CreateLevelMenu	42
RacingGameBot.Menu.EndMenu	45
RacingGameBot.Menu.MainMenu	46
RacingGameBot.Menu.PauseMenu	48
RacingGameBot.Play.CarAgent	49
RacingGameBot.Play.MultiTraining	51
RacingGameBot.Terrains.Loop	51
RacingGameBot.Terrains.OrientedPoint	54
RacingGameBot.Terrains.TerrainGenerator	55
RacingGameBot.Terrains.TerrainLoader	58
RacingGameBot.Tests.BezierTests	60
RacingGameBot.Tests.TextureTests	61
RacingGameBot.Utils.Objects	65



B.4 Namespace Documentation

B.4.1 RacingGameBot.Data Namespace Reference

Classes

class [TerrainGenData](#)

class [UpdatableData](#)

Enumerations

enum [TerrainType](#) { [Basic](#), [Forest](#), [Desert](#), [Mountains](#) }

B.4.2 RacingGameBot.Editors Namespace Reference

Classes

class [BuildScript](#)

class [TerrainGeneratorEditor](#)

class [TerrainLoaderEditor](#)

class [UpdatableDataEditor](#)

B.4.3 RacingGameBot.Menu Namespace Reference

Classes

class [ButtonActions](#)

class [CreateLevelMenu](#)

class [EndMenu](#)

class [MainMenu](#)

class [PauseMenu](#)

B.4.4 RacingGameBot.Play Namespace Reference

Classes

class [CarAgent](#)

class [MultiTraining](#)



B.4.5 RacingGameBot.Terrains Namespace Reference

Classes

```
class Loop  
class OrientedPoint  
class TerrainGenerator  
class TerrainLoader
```

B.4.6 RacingGameBot.Tests Namespace Reference

Classes

```
class BezierTests  
class TextureTests
```

Enumerations

```
enum MyTexture { Grass1, Grass2, Grass3, Road1,  
    Road2, Road3, Road4, Rocks1,  
    Rocks2, Rocks3, Rocks4, Sand1,  
    Snow1, Snow2, Snow3, Water1 }
```

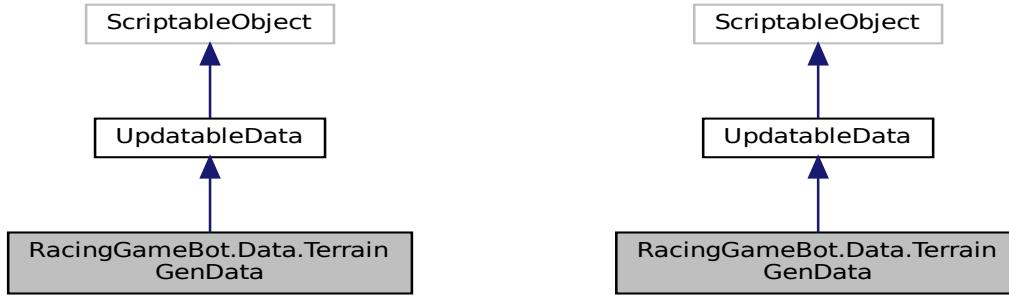
B.4.7 RacingGameBot.Utils Namespace Reference

Classes

```
class Bezier  
class Logging  
class Objects  
class Parser  
class Variables
```

B.5 Class Documentation

B.5.1 RacingGameBot.Data.TerrainGenData Class Reference



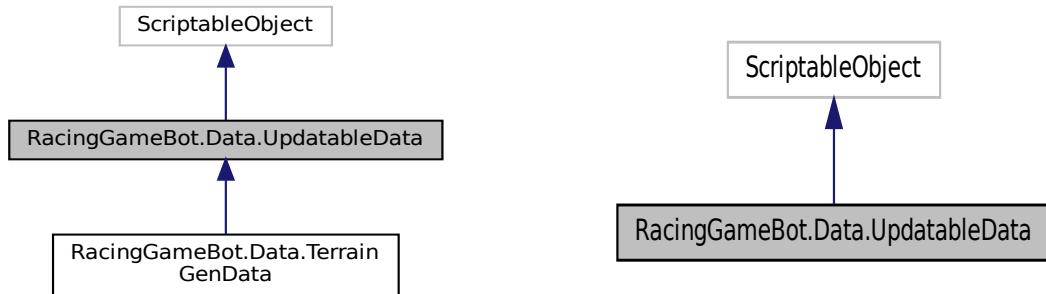
Rysunek B.1: Inheritance diagram for
RacingGameBot.Data.TerrainGenData

Rysunek B.2: Collaboration diagram for
RacingGameBot.Data.TerrainGenData

Public Attributes

```
TerrainType terrainType = TerrainType.Forest  
float roadLength = 10000f  
float roadWidth = 5f  
float paddingPercent = 0.2f  
int numberOfSegments = 4  
int numberOfCheckpoints = 40  
float terrainDetailsMain = 0.3f  
float terrainDetailsMinor = 0.05f  
float terrainDetailsTiny = 0f  
float terrainScaleMain = 1.5f  
float terrainScaleMinor = 5f  
float terrainScaleTiny = 5f  
float offsetX = 0f  
float offsetY = 0f
```

B.5.2 RacingGameBot.Data.UpdatableData Class Reference



Rysunek B.3: Inheritance diagram for
RacingGameBot.Data.UpdatableData

Rysunek B.4: Collaboration diagram for
RacingGameBot.Data.UpdatableData

Public Member Functions

```
void NotifyOfUpdatedValues ()
```

Public Attributes

```
bool autoUpdate = true
```

Protected Member Functions

```
virtual void OnValidate ()
```

B.6 RacingGameBot.Editors.BuildScript Class Reference

Static Public Member Functions

```
static void BuildGameLinux ()
```

Build game executable for Linux

```
static void BuildGameWin ()
```

Build game executable for Windows

B.6.1 Member Function Documentation

BuildGameLinux()

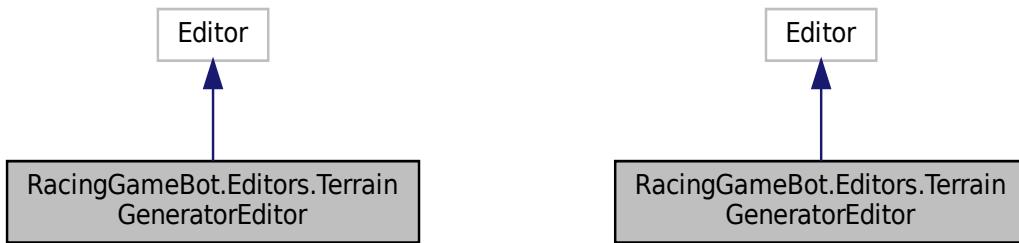
```
static void RacingGameBot.Editors.BuildScript.BuildGameLinux ()  
Build game executable for Linux
```



BuildGameWin()

```
static void RacingGameBot.Editors.BuildScript.BuildGameWin ()  
Build game executable for Windows
```

B.6.2 RacingGameBot.Editors.TerrainGeneratorEditor Class Reference



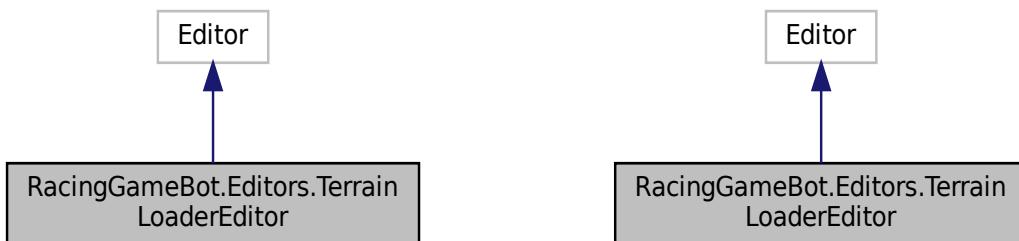
Rysunek B.5: Inheritance diagram for
RacingGameBot.Editors.TerrainGeneratorEditor

Rysunek B.6: Collaboration diagram for
RacingGameBot.Editors.TerrainGeneratorEditor

Public Member Functions

```
override void OnInspectorGUI ()
```

B.6.3 RacingGameBot.Editors.TerrainLoaderEditor Class Reference



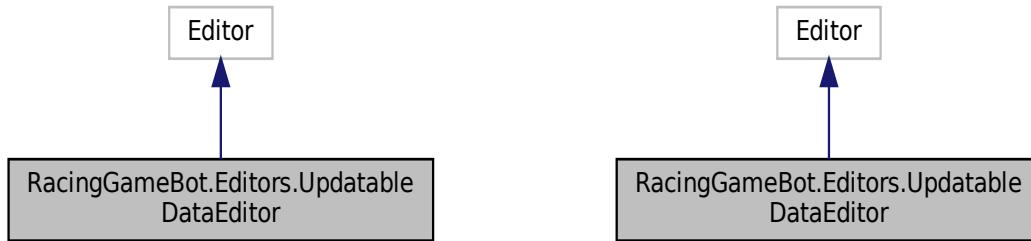
Rysunek B.7: Inheritance diagram for
RacingGameBot.Editors.TerrainLoaderEditor

Rysunek B.8: Collaboration diagram for
RacingGameBot.Editors.TerrainLoaderEditor

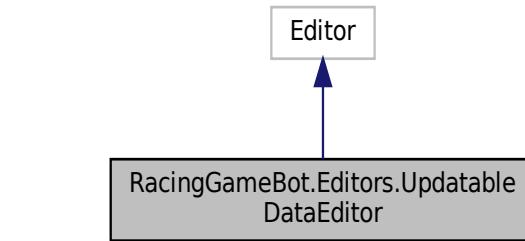
Public Member Functions

```
override void OnInspectorGUI ()
```

B.6.4 RacingGameBot.Editors.UpdatableDataEditor Class Reference



Rysunek B.9: Inheritance diagram for
RacingGameBot.Editors.UpdatableDataEditor



Rysunek B.10: Collaboration diagram for
RacingGameBot.Editors.UpdatableDataEditor

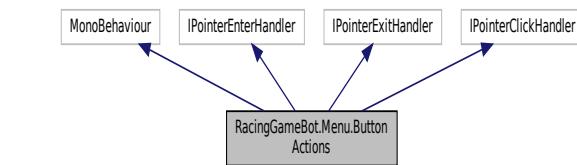
Public Member Functions

```
override void OnInspectorGUI ()
```

B.6.5 RacingGameBot.Menu.ButtonActions Class Reference



Rysunek B.11: Inheritance diagram for
RacingGameBot.Menu.ButtonActions



Rysunek B.12: Collaboration diagram for
RacingGameBot.Menu.ButtonActions

Public Member Functions

```
void OnPointerEnter (PointerEventData eventData)
```

Change text size and color on hover enter

```
void OnPointerExit (PointerEventData eventData)
```



Change text size and color on hover exit

```
void OnPointerClick (PointerEventData eventData)
```

Change text size and color on click

B.6.6 Member Function Documentation

OnPointerClick()

```
void RacingGameBot.Menu.ButtonActions.OnPointerClick (
    [PointerEventData] eventData
)
```

Change text size and color on click

eventData	Click event data
-----------	------------------

OnPointerEnter()

```
void RacingGameBot.Menu.ButtonActions.OnPointerEnter (
    [PointerEventData] eventData
)
```

Change text size and color on hover enter

eventData	Hover event data
-----------	------------------

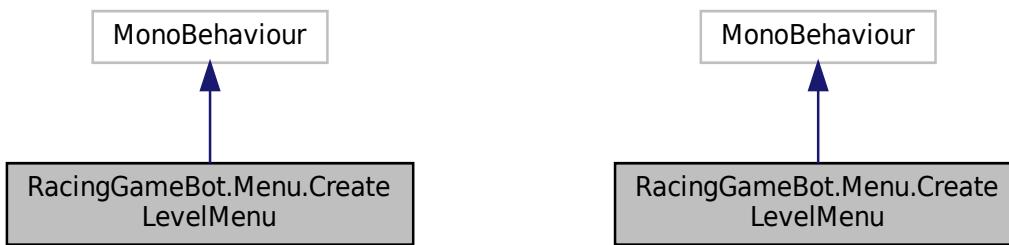
OnPointerExit()

```
void RacingGameBot.Menu.ButtonActions.OnPointerExit (
    [PointerEventData] eventData
)
```

Change text size and color on hover exit

eventData	Hover event data
-----------	------------------

B.6.7 RacingGameBot.Menu.CreateLevelMenu Class Reference



Rysunek B.13: Inheritance diagram for
RacingGameBot.Menu.CreateLevelMenu

Rysunek B.14: Collaboration diagram for
RacingGameBot.Menu.CreateLevelMenu



Public Member Functions

```
void Rotate (float angle)
Rotate terrain preview

void MenuOptionChangeComplexity (float value)
Change road complexity

void MenuOptionChangeDetailsMain (float value)

void MenuOptionChangeDetailsMinor (float value)

void MenuOptionChangeDetailsTiny (float value)

void MenuOptionChangeScaleMain (float value)

void MenuOptionChangeScaleMinor (float value)

void MenuOptionChangeScaleTiny (float value)

void MenuOptionChangeOffsetX (float value)

void MenuOptionChangeOffsetY (float value)

void MenuOptionChangeTerrainType (int value)

void MenuOptionChangeSeed (string value)
Change road generation seed

void MenuOptionSetFilename (string value)
Change save filename

void Save ()
Save generated terrain

void SaveCallbackUpdateTime (int percentage)
Update saving progress

void SaveCallbackEnd ()
Open main menu after finished saving
```

Public Attributes

```
GameObject terrainGO

GameObject cameraGO

GameObject panelGO

GameObject loadingGO
```

B.6.8 Member Function Documentation

**MenuOptionChangeComplexity()**

```
void RacingGameBot.Menu.CreateLevelMenu.MenuOptionChangeComplexity (
    [float] value
)
```

Change road complexity

value	Complexity
-------	------------

MenuOptionChangeSeed()

```
void RacingGameBot.Menu.CreateLevelMenu.MenuOptionChangeSeed (
    [string] value
)
```

Change road generation seed

value	Seed
-------	------

MenuOptionSetFilename()

```
void RacingGameBot.Menu.CreateLevelMenu.MenuOptionSetFilename (
    [string] value
)
```

Change save filename

value	Complexity
-------	------------

Rotate()

```
void RacingGameBot.Menu.CreateLevelMenu.Rotate (
    [string] value
)
```

Rotate terrain preview

angle	Angle
-------	-------

Save()

```
void RacingGameBot.Menu.CreateLevelMenu.Save (
    [string] value
)
```

Save generated terrain

SaveCallbackEnd()

```
void RacingGameBot.Menu.CreateLevelMenu.SaveCallbackEnd (
    [string] value
)
```

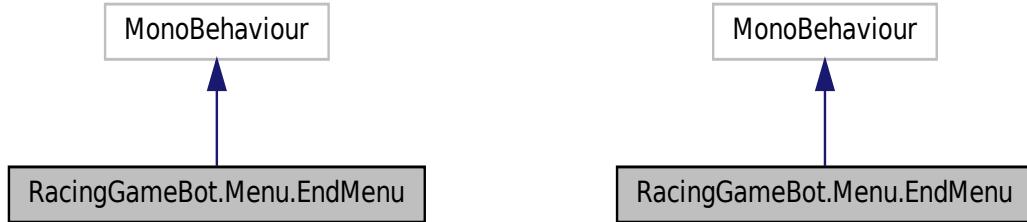
Open main menu after finished saving

SaveCallbackUpdateTime()

```
void RacingGameBot.Menu.CreateLevelMenu.SaveCallbackUpdateTime (
    [string] value
)
```

Update saving progress

B.6.9 RacingGameBot.Menu.EndMenu Class Reference



Rysunek B.15: Inheritance diagram for
RacingGameBot.Menu.EndMenu

Rysunek B.16: Collaboration diagram for
RacingGameBot.Menu.EndMenu

Public Member Functions

void **Show** (bool won)

Show menu at the end of race

void **QuitGame** ()

Quit game and return to main menu

Public Attributes

GameObject **endMenu**

GameObject **winningText**

B.6.10 Member Function Documentation

QuitGame()

```
void RacingGameBot.Menu.EndMenu.QuitGame ()  
Quit game and return to main menu
```

Show()

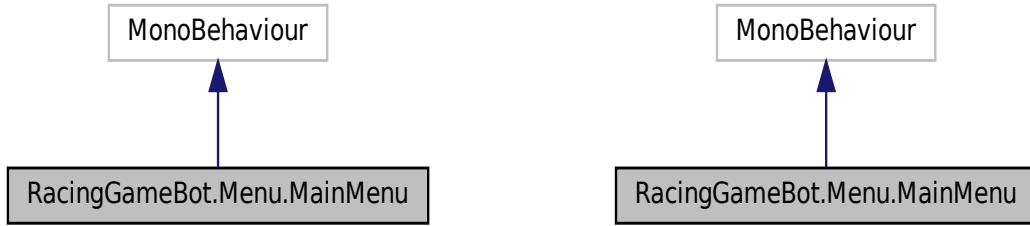
```
void RacingGameBot.Menu.EndMenu.Show (  
    [bool] won  
)
```

Show menu at the end of race

won	Whether user won or lost
-----	--------------------------



B.6.11 RacingGameBot.Menu.MainMenu Class Reference



Rysunek B.17: Inheritance diagram for
RacingGameBot.Menu.MainMenu

Rysunek B.18: Collaboration diagram for
RacingGameBot.Menu.MainMenu

Public Member Functions

```

void MenuOptionPlayGame ()
    Open play screen
void MenuOptionCreateLevel ()
    Open level creation screen
void MenuOptionQuitGame ()
    Quit game
void MenuOptionMoveToPlayMenu ()
    Open play level choosing screen
void MenuOptionMoveToMainMenu ()
    Get back from level choosing screen to main screen
void MenuOptionChooseFilename (int file)
    Select level name

```

Public Attributes

```

string filename
List< TMPro.TMP_Dropdown.OptionData > options
GameObject mainMenu
GameObject playMenu

```

B.6.12 Member Function Documentation

**MenuOptionChooseFilename()**

```
void RacingGameBot.Menu.MainMenu.MenuOptionChooseFilename (
    [int] file
```

Select level name

<i>file</i>	Number of file on the list
-------------	----------------------------

MenuOptionCreateLevel()

```
void RacingGameBot.Menu.MainMenu.MenuOptionCreateLevel ()  
Open level creation screen
```

MenuOptionMoveToMainMenu()

```
void RacingGameBot.Menu.MainMenu.MenuOptionMoveToMainMenu ()  
Get back from level choosing screen to main screen
```

MenuOptionMoveToPlayMenu()

```
void RacingGameBot.Menu.MainMenu.MenuOptionMoveToPlayMenu ()  
Open play level choosing screen
```

MenuOptionPlayGame()

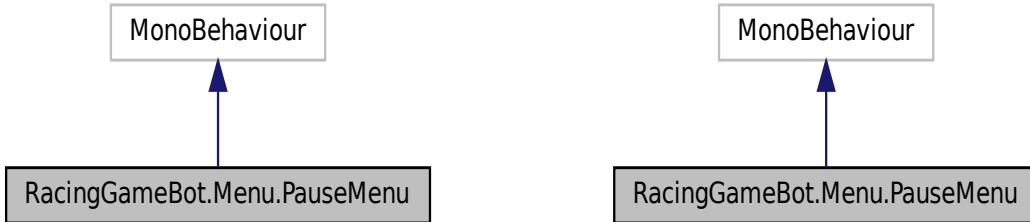
```
void RacingGameBot.Menu.MainMenu.MenuOptionPlayGame ()  
Open play screen
```

MenuOptionQuitGame()

```
void RacingGameBot.Menu.MainMenu.MenuOptionQuitGame ()  
Quit game
```



B.6.13 RacingGameBot.Menu.PauseMenu Class Reference



Rysunek B.19: Inheritance diagram for
RacingGameBot.Menu.PauseMenu

Rysunek B.20: Collaboration diagram for
RacingGameBot.Menu.PauseMenu

Public Member Functions

```

void ResumeGame ()
    Hide menu and resume game
void PauseGame ()
    Show menu and pause game
void QuitGame ()
    Quit game and open main menu

```

Public Attributes

`GameObject pauseMenu`

Static Public Attributes

`static bool gameIsPaused = false`

B.6.14 Member Function Documentation

PauseGame()

```

void RacingGameBot.Menu.PauseMenu.PauseGame ()
Show menu and pause game

```

QuitGame()

```

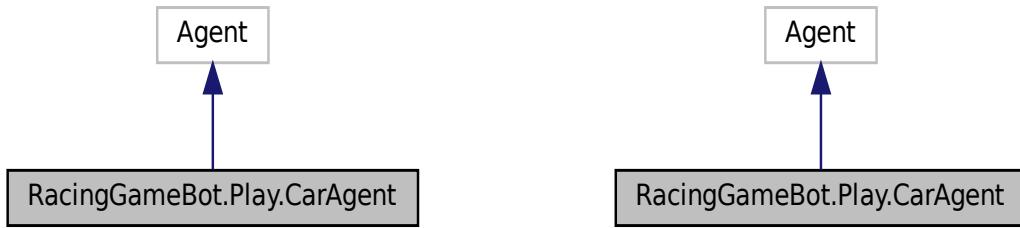
void RacingGameBot.Menu.PauseMenu.QuitGame ()
Quit game and open main menu

```

ResumeGame()

```
void RacingGameBot.Menu.PauseMenu.ResumeGame ()  
Hide menu and resume game
```

B.6.15 RacingGameBot.Play.CarAgent Class Reference



Rysunek B.21: Inheritance diagram for
RacingGameBot.Play.CarAgent

Rysunek B.22: Collaboration diagram for
RacingGameBot.Play.CarAgent

Public Member Functions

override void **OnEpisodeBegin** ()

Set car starting position in first episode, reset it at the beginning of each episode

override void **CollectObservations** (VectorSensor sensor)

Collect observation data from environment

override void **OnActionReceived** (ActionBuffers actions)

Move car based on inference results and assign rewards

void **OnTriggerEnter** (Collider other)

Handle car collisions with environment objects, like other cars, checkpoints, finish, etc.

override void **Heuristic** (in ActionBuffers actionsOut)

Read user inputs for manual steering

Public Attributes

bool showGizmos = false

bool playMode = false

B.6.16 Member Function Documentation

**CollectObservations()**

```
override void RacingGameBot.Play.CarAgent.CollectObservations (
    [VectorSensor] sensor
```

)

Collect observation data from environment:

- distance to road center
- current input axis positions for forward and sideways movement
- slope of terrain
- velocity

<i>sensor</i>	Car sensor
---------------	------------

Heuristic()

```
override void RacingGameBot.Play.CarAgent.Heuristic (
    [in ActionBuffers] actionsOut
```

)

Read user inputs for manual steering

<i>actionsOut</i>	Car actions
-------------------	-------------

OnActionReceived()

```
override void RacingGameBot.Play.CarAgent.OnActionReceived (
    [ActionBuffers] actions
```

)

Move car based on inference results and assign rewards

<i>actions</i>	Received actions
----------------	------------------

OnEpisodeBegin()

```
override void RacingGameBot.Play.CarAgent.OnEpisodeBegin()
```

Set car starting position in first episode, reset it at the beginning of each episode

OnTriggerEnter()

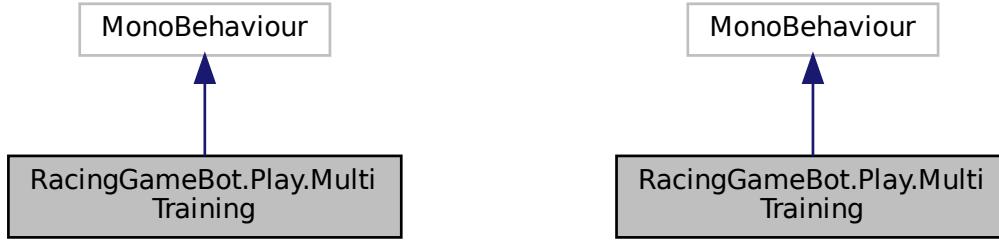
```
void RacingGameBot.Play.CarAgent.OnTriggerEnter (
    [Collider] other
```

)

Handle car collisions with environment objects, like other cars, checkpoints, finish, etc.

<i>other</i>	Object with which car collided
--------------	--------------------------------

B.6.17 RacingGameBot.Play.MultiTraining Class Reference



Rysunek B.23: Inheritance diagram for RacingGameBot.Play.MultiTraining

Rysunek B.24: Collaboration diagram for RacingGameBot.Play.MultiTraining

Public Attributes

```
bool playMode = false  
bool showGizmos = false  
string filenames = ""  
int numberofInstances = 1
```

B.6.18 RacingGameBot.Terrains.Loop Class Reference

Public Member Functions

`Loop ()`

Create simple circular loop of size (1, 1)

`Loop (Vector3 parentPosition, List< OrientedPoint > _points, Data.TerrainGenData _terrainGenData)`

Create loop from saved data

`Loop (Vector3 parentPosition, Data.TerrainGenData _terrainGenData)`

Create loop based of additional params

`float GetHeight (float x, float z)`

Get height for each point

`OrientedPoint GetCentralPoint (int i)`

Get starting point of each segment

`List< OrientedPoint > GetSegmentBezierPoints (int i)`

Get bezier points for ith segment



`OrientedPoint GetNearestBezierPoint (Vector3 target)`

Calculate what is the nearest point on created loop

`List< float > GetBorder ()`

Find minimal and maximal values of created loop

`int LoopIndex (int i)`

Make sure index is always in the boundaries

`int LoopCentralIndex (int i)`

Make sure index is always in the boundaries of segments

`List< OrientedPoint > GetEquallySpacedPoints (int n)`

Get points along loop, divided equally

`void Translate (Vector3 pos)`

Move each point of loop by some vector

Public Attributes

`float minDistanceBetweenPoints = 0.1f`

`List< OrientedPoint > points`

`Data.TerrainGenData terrainGenData`

Properties

`int NumberOfSegments [get]`

B.6.19 Constructor & Destructor Documentation

Loop() [1/3]

`RacingGameBot.Terrains.Loop.Loop ()`

Create simple circular loop of size (1, 1)

Loop() [2/3]

```
RacingGameBot.Terrains.Loop.Loop (
    [Vector3] parentPosition,
    [List< OrientedPoint >] _points,
    [Data.TerrainGenData] _terrainGenData
)
```

Create loop from saved data

<code>parentPosition</code>	position of loop center
<code>_points</code>	list of loop points
<code>_terrainGenData</code>	terrain data

**Loop()** [3/3]

```
RacingGameBot.Terrains.Loop.Loop (
    [Vector3] parentPosition,
    [Data.TerrainGenData] _terrainGenData
)
```

Create loop based of additional params

<i>parentPosition</i>	position of loop center
<i>_terrainGenData</i>	terrain data

B.6.20 Member Function Documentation

GetBorder()

```
List<float> RacingGameBot.Terrains.Loop.GetBorder ()
```

Find minimal and maximal values of created loop

return minX, maxX, minZ, maxZ

GetCentralPoint()

```
OrientedPoint RacingGameBot.Terrains.Loop.GetCentralPoint (
    [int] i
)
```

Get starting point of each segment

<i>i</i>	Number of segment from created loop
----------	-------------------------------------

return Starting point

GetEquallySpacedPoints()

```
List<OrientedPoint> RacingGameBot.Terrains.Loop.GetEquallySpacedPoints (
    [int] n
)
```

Get points along loop, divided equally

<i>n</i>	Number of points
----------	------------------

return List of points

GetHeight()

```
float RacingGameBot.Terrains.Loop.GetHeight (
    [float] x,
    [float] z
)
```

Get height for each point

<i>x</i>	X position
<i>z</i>	Z position

return height

GetNearestBezierPoint()

```
OrientedPoint RacingGameBot.Terrains.Loop.GetNearestBezierPoint (
    [Vector3] target
)
```



)
Calculate what is the nearest point on created loop

<code>target</code>	From what point to calculate distance
---------------------	---------------------------------------

return Quaternion where xyz is point and w is distance

GetSegmentBezierPoints()

```
List<OrientedPoint> RacingGameBot.Terrains.Loop.GetSegmentBezierPoints (
```

[int]	<code>i</code>
-------	----------------

)

Get bezier points for i-th segment

<code>i</code>	Number of segment from created loop
----------------	-------------------------------------

return List of 2 end points and 2 controls

LoopCentralIndex()

```
int RacingGameBot.Terrains.Loop.LoopCentralIndex (
```

[int]	<code>i</code>
-------	----------------

)

Make sure index is always in the boundaries of segments

<code>i</code>	Index
----------------	-------

return Looped index

LoopIndex()

```
int RacingGameBot.Terrains.Loop.LoopIndex (
```

[int]	<code>i</code>
-------	----------------

)

Make sure index is always in the boundaries

<code>i</code>	Index
----------------	-------

return Looped index

Translate()

```
void RacingGameBot.Terrains.Loop.Translate (
```

[Vector3]	<code>pos</code>
-----------	------------------

)

Move each point of loop by some vector

<code>pos</code>	Move vector
------------------	-------------

return height

B.6.21 RacingGameBot.Terrains.OrientedPoint Class Reference

Public Member Functions

OrientedPoint (Vector3 position)

OrientedPoint (Vector3 position, Quaternion rotation)

OrientedPoint (Vector3 position, Quaternion rotation, float other)

OrientedPoint (Vector3 position, Vector3 forward)

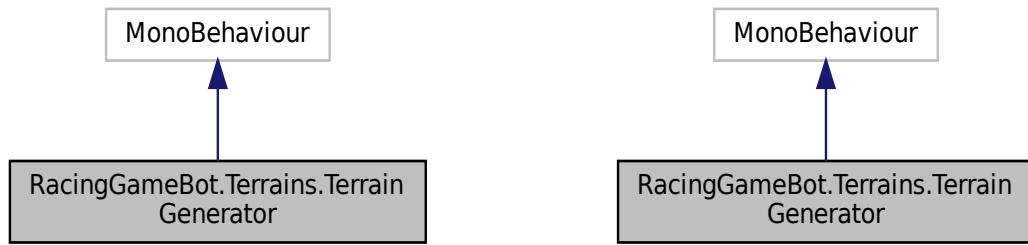
override string **ToString** ()

Vector3 **LocalToWorldPosition** (Vector3 localSpaceposition)

Public Attributes

```
Vector3 position  
Quaternion rotation  
float other
```

B.6.22 RacingGameBot.Terrains.TerrainGenerator Class Reference



Rysunek B.25: Inheritance diagram for
RacingGameBot.Terrains.TerrainGenerator

Rysunek B.26: Collaboration diagram for
RacingGameBot.Terrains.TerrainGenerator

Public Member Functions

```
void ShowRoadBezier ()  
    Draw road in editor  
void GenerateTerrain ()  
    Generate terrain  
void GenerateLoop ()  
    Generate road loop  
void Save ()  
    Save current terrain  
void GenerateTerrainShape ()  
    Create terrain shape (heightmap) based on perlin noise  
void RemoveTerrainTextures ()  
    Clear terrain textures  
IEnumerator GenerateTerrainTextures (System.Action< int > callbackUpdateTime=null, System.Action  
callbackEnd=null)  
    Add textures and Utils.Objects to terrain based on terrain type
```



Static Public Member Functions

static TerrainLayer [GetTerrainLayerTexture](#) (string textureName, float size)

Load texture based on name

static TerrainLayer [GetTerrainLayerColor](#) (Color color)

Load single-colored texture

Public Attributes

bool **generateOnChanges** = false

int **seed** = 0

[Data.TerrainGenData](#) **terrainGenData**

bool **showCheckpoints** = false

bool **showBorders** = false

bool **showRoadBezier** = false

string **filename**

B.6.23 Member Function Documentation

GenerateLoop()

```
void RacingGameBot.Terrains.TerrainGenerator.GenerateLoop ()  
Generate road loop
```

GenerateTerrain()

```
void RacingGameBot.Terrains.TerrainGenerator.GenerateTerrain ()  
Generate terrain
```

GenerateTerrainShape()

```
void RacingGameBot.Terrains.TerrainGenerator.GenerateTerrainShape ()  
Create terrain shape (heightmap) based on perlin noise
```

GenerateTerrainTextures()

```
IEnumerator RacingGameBot.Terrains.TerrainGenerator.GenerateTerrainTextures (  
    [System.Action< int >] callbackUpdateTime = null,  
    [System.Action] callbackEnd = null  
)
```

Add textures and objects to terrain based on terrain type



GetTerrainLayerColor()

```
static TerrainLayer RacingGameBot.Terrains.TerrainGenerator.GetTerrainLayerColor (
    [Color] color
)
Load single-colored texture


|              |               |
|--------------|---------------|
| <i>color</i> | Texture color |
|--------------|---------------|


return TerrainLayer object with loaded color
```

GetTerrainLayerTexture()

```
static TerrainLayer RacingGameBot.Terrains.TerrainGenerator.GetTerrainLayerTexture (
    [string] textureName,
    [float] size
)
Load texture based on name


|                    |                      |
|--------------------|----------------------|
| <i>textureName</i> | Name of texture file |
| <i>size</i>        | Texture tiling size  |


return TerrainLayer object with loaded texture
```

RemoveTerrainTextures()

```
void RacingGameBot.Terrains.TerrainGenerator.RemoveTerrainTextures ()  
Clear terrain textures
```

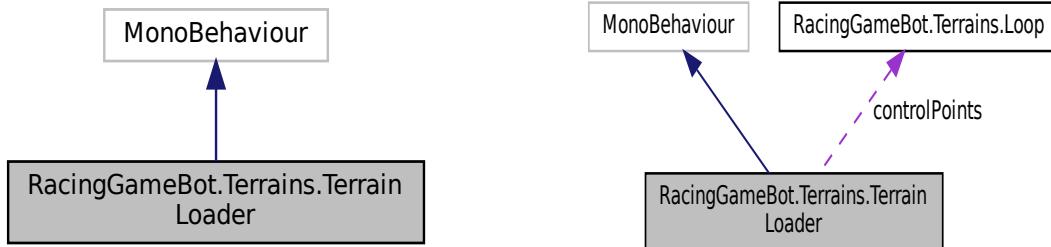
Save()

```
void RacingGameBot.Terrains.TerrainGenerator.Save ()  
Save current terrain
```

ShowRoadBezier()

```
void RacingGameBot.Terrains.TerrainGenerator.ShowRoadBezier ()  
Draw road in editor
```

B.6.24 RacingGameBot.Terrains.TerrainLoader Class Reference



Rysunek B.27: Inheritance diagram for
RacingGameBot.Terrains.TerrainLoader

Rysunek B.28: Collaboration diagram for
RacingGameBot.Terrains.TerrainLoader

Public Member Functions

```

void LoadTerrain (string filename, bool _playMode=true)
    Load saved terrain from file
void LoadData (string filename)
    Load Terrain object from file
void LoadObjects (string filename)
    Load other objects (cars, borders, checkpoints, etc) from file

```

Static Public Member Functions

```

static void SaveTerrain (TerrainData terrainData, Data.TerrainGenData terrainGenData, OrientedPoint[] controlPoints, Vector3 meta, OrientedPoint[] cars, Vector3 carSize, OrientedPoint[] checkpoints, Vector3 checkpointSize, string filename)
    Save generated terrain into file

```

Public Attributes

```

string basePath
Data.TerrainGenData terrainGenData
Loop controlPoints
GameObject startFinish
List< GameObject > checkpoints
List< GameObject > cars
GameObject terrainGO
Vector3 terrainDataSize

```



```
bool showCheckpoints = false  
bool showBorders = false  
bool showGizmos = false  
bool playMode = false  
string filename
```

B.6.25 Member Function Documentation

LoadData()

```
void RacingGameBot.Terrains.TerrainLoader.LoadData (  
    [string] filename  
)  
Load Terrain object from file
```

LoadObjects()

```
void RacingGameBot.Terrains.TerrainLoader.LoadObjects (  
    [string] filename  
)  
Load other objects (cars, borders, checkpoints, etc) from file
```

LoadTerrain()

```
void RacingGameBot.Terrains.TerrainLoader.LoadTerrain (  
    [string] filename,  
    [bool] playMode = true  
)  
Load saved terrain from file
```

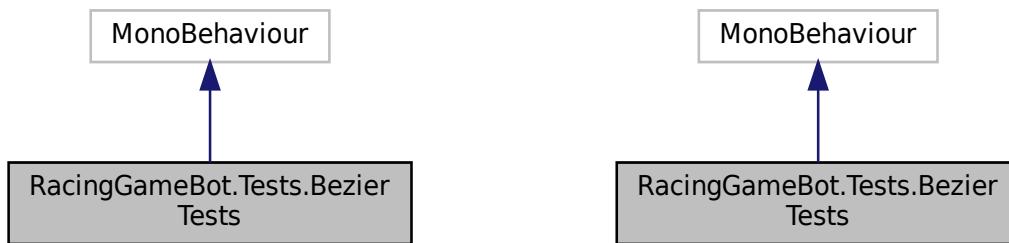
SaveTerrain()

```
static void RacingGameBot.Terrains.TerrainLoader.SaveTerrain (  
    [TerrainData] terrainData,  
    [Data.TerrainGenData] terrainGenData,  
    [OrientedPoint[]] controlPoints,  
    [Vector3] meta,  
    [OrientedPoint[]] cars,  
    [Vector3] carSize,  
    [OrientedPoint[]] checkpoints,  
    [Vector3] checkpointSize,  
    [string] filename
```

)
Save generated terrain into file

<i>terrainData</i>	Data of Unity Terrain object
<i>terrainGenData</i>	Terrain parameters
<i>controlPoints</i>	Loop points
<i>meta</i>	position of finish object
<i>cars</i>	list of positions of cars
<i>carSize</i>	size of car
<i>checkpoints</i>	list of positions of checkpoints
<i>checkpointSize</i>	size of checkpoint
<i>filename</i>	name od save file

B.6.26 RacingGameBot.Tests.BezierTests Class Reference



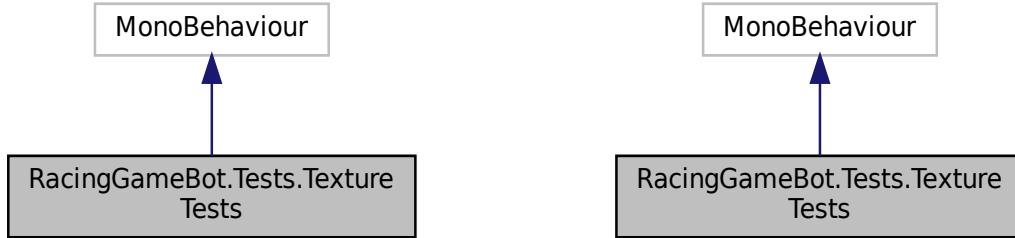
Rysunek B.29: Inheritance diagram for
RacingGameBot.Tests.BezierTests

Rysunek B.30: Collaboration diagram for
RacingGameBot.Tests.BezierTests

Public Attributes

GameObject **sphere**

B.6.27 RacingGameBot.Tests.TextureTests Class Reference



Rysunek B.31: Inheritance diagram for
RacingGameBot.Tests.TextureTests

Rysunek B.32: Collaboration diagram for
RacingGameBot.Tests.TextureTests

Public Attributes

bool **showAll**
Terrain **terrain**
MyTexture **texture**

B.7 RacingGameBot.Utils.Bezier Class Reference

Static Public Member Functions

static Vector3 [EvaluateQuadratic](#) (Vector3 a, Vector3 b, Vector3 c, float t)
Get point at t% along quadratic bezier curve

static Vector3 [EvaluateCubic](#) (Vector3 a, Vector3 b, Vector3 c, Vector3 d, float t)
Get point at t% along cubic bezier curve

static float [EvaluateCubicLength](#) (Vector3 a, Vector3 b, Vector3 c, Vector3 d)
Evaluate length of bezier curve

static Terrains.OrientedPoint [GetBezierOrientedPoint](#) (Vector3 a, Vector3 b, Vector3 c, Vector3 d, float t)
Get oriented point at t% along bezier curve

static Terrains.OrientedPoint [GetNearestBezierPointOld](#) (Vector3 bezierA, Vector3 bezierB, Vector3 bezierC, Vector3 bezierD, Vector3 target)
Calculate what is the nearest point on bezier cubic curve using method 1

static Terrains.OrientedPoint [GetNearestBezierPoint](#) (Vector3 bezierA, Vector3 bezierB, Vector3 bezierC, Vector3 bezierD, Vector3 target)
Calculate what is the nearest point on bezier cubic curve using method 2



B.7.1 Member Function Documentation

EvaluateCubic()

```
static Vector3 RacingGameBot.Utils.Bezier.EvaluateCubic (
    [Vector3] a,
    [Vector3] b,
    [Vector3] c,
    [Vector3] d,
    [float] t
)
```

Get point at t% along cubic bezier curve

<i>a</i>	Bezier curve start
<i>b</i>	Bezier curve control 1
<i>c</i>	Bezier curve control 2
<i>d</i>	Bezier curve end
<i>t</i>	Percent of curve distance

return Point at t% along cubic bezier curve

EvaluateCubicLength()

```
static float RacingGameBot.Utils.Bezier.EvaluateCubicLength (
    [Vector3] a,
    [Vector3] b,
    [Vector3] c,
    [Vector3] d
)
```

Evaluate length of bezier curve

<i>a</i>	Bezier curve start
<i>b</i>	Bezier curve control 1
<i>c</i>	Bezier curve control 2
<i>d</i>	Bezier curve end

return Evaluated length

EvaluateQuadratic()

```
static Vector3 RacingGameBot.Utils.Bezier.EvaluateQuadratic (
    [Vector3] a,
    [Vector3] b,
    [Vector3] c,
    [float] t
)
```

Get point at t% along quadratic bezier curve

<i>a</i>	Bezier curve start
<i>b</i>	Bezier curve control 1
<i>c</i>	Bezier curve control 2
<i>t</i>	Percent of curve distance

return Point at t% along quadratic bezier curve

**GetBezierOrientedPoint()**

```
static Terrains.OrientedPoint RacingGameBot.Utils.Bezier.GetBezierOrientedPoint (
    [Vector3] a,
    [Vector3] b,
    [Vector3] c,
    [Vector3] d,
    [float] t
)
```

Get oriented point at t% along bezier curve

a	Bezier curve start
b	Bezier curve control 1
c	Bezier curve control 2
d	Bezier curve end
t	Percent of curve distance

return Oriented point at t% along bezier curve

GetNearestBezierPoint()

```
static Terrains.OrientedPoint RacingGameBot.Utils.Bezier.GetNearestBezierPoint (
    [Vector3] bezierA,
    [Vector3] bezierB,
    [Vector3] bezierC,
    [Vector3] bezierD,
    [Vector3] target
)
```

Calculate what is the nearest point on bezier cubic curve using method 2

bezierA	Bezier curve start
bezierB	Bezier curve control 1
bezierC	Bezier curve control 2
bezierD	Bezier curve end
target	From what point to calculate distance

return OrientedPoint with position, rotation and distance

GetNearestBezierPointOld()

```
static Terrains.OrientedPoint RacingGameBot.Utils.Bezier.GetNearestBezierPointOld (
    [Vector3] bezierA,
    [Vector3] bezierB,
    [Vector3] bezierC,
    [Vector3] bezierD,
    [Vector3] target
)
```

Calculate what is the nearest point on bezier cubic curve using method 1

bezierA	Bezier curve start
bezierB	Bezier curve control 1
bezierC	Bezier curve control 2
bezierD	Bezier curve end
target	From what point to calculate distance

return OrientedPoint with position, rotation and distance



B.8 RacingGameBot.Utils.Logging Class Reference

Static Public Member Functions

`static string GetListString< T > (List< T > l)`

Convert list to string

`static string GetParamString< T > (params T[] args)`

Convert multiple params to one string

`static void LogToFile (string filename, string text)`

Append string to file. Create file if not exists.

B.8.1 Member Function Documentation

`GetListString< T >()`

```
static string RacingGameBot.Utils.Logging.GetListString< T > (
    [List< T >] l
)
```

Convert list to string

<code>l</code>	list of objects of type T
----------------	---------------------------

return list in string format

`GetParamString< T >()`

```
static string RacingGameBot.Utils.Logging.GetParamString< T > (
    [params T[]] args
)
```

Convert multiple params to one string

<code>args</code>	objects of type T
-------------------	-------------------

return objects in one string

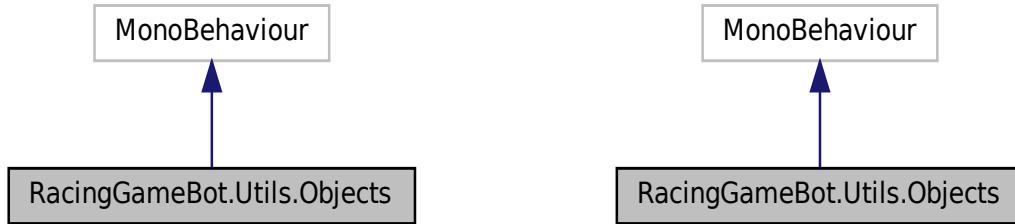
`LogFile()`

```
static void RacingGameBot.Utils.Logging.LogToFile (
    [string] filename,
    [string] text
)
```

Append string to file. Create file if not exists.

<code>filename</code>	name of log file
<code>text</code>	data to log

B.8.2 RacingGameBot.Utils.Objects Class Reference



Rysunek B.33: Inheritance diagram for
RacingGameBot.Utils.Objects

Rysunek B.34: Collaboration diagram for
RacingGameBot.Utils.Objects

Static Public Member Functions

```
static void RemoveAllObjectsByTag (string tag)
    Remove all objects by tag
static void RemoveObjectsByTagInParent (string tag, GameObject parent)
    Remove all objects by tag, which are children of some object
static GameObject PutParentObject (string tag, string name)
    Create empty object with tag and name
static GameObject PutObject (string prefabName, string tag, string name, Terrains.OrientedPoint
    posrot, Vector3? size=null)
    Create object from prefab
static GameObject GetChildWithName (GameObject obj, string name)
    Get object by name, which is below some object in hierarchy
static GameObject GetParentWithName (GameObject obj, string name)
    Get object by name, which is above some object in hierarchy
```

B.8.3 Member Function Documentation

GetChildWithName()

```
static GameObject RacingGameBot.Utils.Objects.GetChildWithName (
    [GameObject]obj,
    [string]name
)
```

Get object by name, which is below some object in hierarchy

<i>obj</i>	Parent object
<i>name</i>	Object name

**GetParentWithName()**

```
static GameObject RacingGameBot.Utils.Objects.GetParentWithName (
    [GameObject]obj,
    [string]name
)
```

Get object by name, which is above some object in hierarchy

<i>obj</i>	Child object
<i>name</i>	Object name

PutObject()

```
static GameObject RacingGameBot.Utils.Objects.PutObject (
    [string] prefabName,
    [string] tag,
    [string] name,
    [Terrains.OrientedPoint]posrot,
    [Vector3?]size = null
)
```

Create object from prefab

<i>prefabName</i>	Name of prefab asset
<i>tag</i>	Object tag
<i>name</i>	Object name
<i>posrot</i>	Position and rotation of object
<i>size</i>	Size of object

return Created GameObject

PutParentObject()

```
static GameObject RacingGameBot.Utils.Objects.PutParentObject (
    [string] tag,
    [string] name
)
```

Create empty object with tag and name

<i>tag</i>	Object tag
<i>name</i>	Object name

return Created GameObject

RemoveAllObjectsByTag()

```
static void RacingGameBot.Utils.Objects.RemoveAllObjectsByTag (
    [string] tag
)
```

Remove all objects by tag

<i>tag</i>	Object tag
------------	------------

RemoveObjectsByTagInParent()

```
static void RacingGameBot.Utils.Objects.RemoveObjectsByTagInParent (
    [string] tag,
    [GameObject] parent
)
```

Remove all objects by tag, which are children of some object

<i>tag</i>	Object tag
<i>parent</i>	Parent object



B.9 RacingGameBot.Utils.Parser Class Reference

Static Public Member Functions

```
static Vector3 Vector3Parse (string sVector)

Parse string to Vector3

static Quaternion QuaternionParse (string sQuaternion)

Parse string to Quaternion

static Terrains.OrientedPoint OrientedPointParse (string sOP)

Parse string to OrientedPoint

static string ColorToString (Color color)

Parse Color to string

static Color ColorParse (string sColor)

Parse string to Color

static Vector3 ToVector3 (Quaternion quaternion)

Convert Quaternion to Vector3, by removing 'w' component
```

B.9.1 Member Function Documentation

ColorParse()

```
static Color RacingGameBot.Utils.Parser.ColorParse (
    [string] sColor
)
```

Parse string to Color

<i>sColor</i>	Color in string format
---------------	------------------------

ColorToString()

```
static string RacingGameBot.Utils.Parser.ColorToString (
    [Color] color
)
```

Parse Color to string

<i>color</i>	Color
--------------	-------

OrientedPointParse()

```
static Terrains.OrientedPoint RacingGameBot.Utils.Parser.OrientedPointParse (
    [string] sOP
)
```

Parse string to OrientedPoint

<i>sOP</i>	OrientedPoint in string format
------------	--------------------------------

**QuaternionParse()**

```
static Quaternion RacingGameBot.Utils.ParserQuaternionParse (
    [string] sQuaternion
)
```

Parse string to Quaternion

<i>sQuaternion</i>	Quaternion in string format
--------------------	-----------------------------

ToVector3()

```
static Vector3 RacingGameBot.Utils.ParserToVector3 (
    [Quaternion] quaternion
)
```

Convert Quaternion to Vector3, by removing 'w' component

<i>quaternion</i>	Quaternion
-------------------	------------

Vector3Parse()

```
static Vector3 RacingGameBot.Utils.ParserVector3Parse (
    [string] sVector
)
```

Parse string to Vector3

<i>sVector</i>	Vector3 in string format
----------------	--------------------------