Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Fyzikální model vozidla v závodní hře

2021

David Kolář

Vedoucí práce:
Mgr. Tomáš Kühr, Ph.D.

Studijní obor: Aplikovaná informatika,
kombinovaná forma

Bibliografické údaje

Autor:	David Kolář
Název práce:	Fyzikální model vozidla v závodní hře
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2021
Studijní obor:	Aplikovaná informatika, kombinovaná forma
Vedoucí práce:	Mgr. Tomáš Kühr, Ph.D.
Počet stran:	30
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	David Kolář
Title:	Car Physics in a Racing Game
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2021
Study field:	Applied Computer Science, combined form
Supervisor:	Mgr. Tomáš Kühr, Ph.D.
Page count:	30
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Práce se zabývá implementací konfigurovatelného jízdního modelu ve videoherním enginu Unity a jeho následným začleněním do herních mechanik. Je popsán celkový proces vývoje, použité komponenty enginu a architektura herní logiky.

Synopsis

The thesis describes the implementation of a configurable driving model in the video game engine Unity and its subsequent integration into game mechanics. The thesis provides analysis of development process, the engine components used, and the architecture of the game logic.

Klíčová slova: Unity; C#; Fyzikální model vozidla; Počítačová hra

Keywords: Unity; C#; Vehicle handling model; Computer game

Rád bych poděkoval Mgr. Tomáši Kührovi, Ph.D. za odborné vedení bakalářské práce.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

1 Úvod

Vývoj videoher se s postupem času stává stále komplexnějším procesem a pro člověka, který se do tohoto procesu snaží vstoupit může být poněkud matoucí, kde začít. Má bakalářská práce bude psaná z pohledu člověka, který má již několikaleté praktické zkušenosti s programováním webových aplikací, ale programování videoher je pro něj v podstatě novinkou.

Jelikož kromě IT jsou mým koníčkem i auta, rozhodl jsem se, že ve videoherním enginu Unity vytvořím vlastní závodní hru, která bude zaměřena na konfigurovatelný jízdní model.

1.1 Zdroj inspirace

Vize byla vytvořit hru s prvky inspirovanými herními sériemi TrackMania[1] a SpinTires[2]. Z první jmenované se mi líbí koncept závodu z bodu A do bodu B v co nejkratším časovém úseku. V druhé hře zase obdivuji propracovaný a hlavně realistický jízdní model.

Co se grafické stránky týče, rozhodl jsem se pro spíše realistické ztvárnění. Nejsem příliš umělecky nadaný a proto jsem se rozhodl použít pro složitější objekty 3D modely dostupné z Unity Asset Store[1].

1.2 Výběr enginu

Pod pojmem videoherní engine se rozumí sada softwarových nástrojů, umožňující snazší a rychlejší vývoj videoher díky tomu, že implementují obecné a znovupoužitelné komponenty, které se ve hrách pravidelně objevují. Existuje mnoho hráčů na tomto trhu a pro začínajícího herního vývojáře může být správný výběr enginu poněkud náročný. Z logických důvodů je dobré hned ze začátku vyškrtnout ty enginy, za které se platí licenční poplatky i při nekomerčním použití. Mezi 3 hlavní kandidáty pro použití v mé bakalářské práci se nakonec dostaly enginy:

- Unity,
- Unreal Engine,
- Godot.

Unreal Engine jsem nepoužil z toho důvodu, že preferuji jazyk C# a pro začínajícího herního vývojáře je tento jazyk snazší na použití. Jinak se při porovnání s Unity jedná o rovnocenné konkurenty, kde rozdíly mezi nimi poznají až profesionální vývojáři.

Godot na druhou stranu programování v C# umožňuje, je ale zatím méně rozšířený a tak se pro něj studijní materiály hledají obtížněji.

[1] https://assetstore.unity.com/

Z těchto důvodů padla finální volba na multiplatformní engine Unity, který podporuje kompilaci jak pro PC a herní konzole, tak mobilní telefony a webové prohlížeče.

1.3 Podmínky použití enginu

V době psaní textu (rok 2021) společnost Unity Technologies poskytuje 4 verze svého enginu.[3] První verze, zvaná Personal Edition je dostupná ke stažení zdarma a uživatel ji může používat i pro komerční projekty, které kumulativně vydělají méně jak 100 000 $.[4] Zbývající 3 verze (Plus, Pro a Enterprise) jsou všechny již placené a liší se mírou technické podpory od společnosti Unity Technoloies a přístupem k některým pokročilým funkcím enginu.

1.4 Učební zdroje

Před tím, než jsem začal vyvíjet samotnou hru, musel jsem projít několik tutoriálů, které mě obeznámily s vývojem v Unity. Především bych chtěl zmínit YouTube kanál Brackeys, na kterém lze najít velké množství návodů pro naprosté začátečníky. Pomohl mi se velmi rychle přenést přes úvodní neznalost prostředí a začít se soustředit na samotný vývoj hry. Dalším hodnotným zdrojem informací byla oficiální dokumentace enginu Unity.

2 Popis základních komponent Unity

Pro přehlednost přidávám tuto kapitolu, aby se čtenář seznámil se základními komponentami enginu Unity, jejich názvy budu v bakalářské práci často používat.

2.1 Scene

Scénou se v Unity myslí základní workspace, do kterého se vkládá herní obsah.[5] Je to prvek, který může obsahovat celou hru nebo pouze její část. Typicky se do separátních scén vkládají jednotlivé levely hry. U menších projektů pak může být vše pouze v jedné scéně. Tento přístup jsem zvolil ve své práci.

2.2 GameObject

GameObject je jedním z nejdůležitějších konceptů v Unity. Každý objekt, který se v Unity nachází (postava, model stromu, zvukový efekt, ...) je definován pomocí třídy *GameObject*.[6] Tyto objekty samy o sobě mnoho funkčnosti neposkytují ale slouží jako kontejner pro tzv. komponenty, které implementují dodatečnou funkcionalitu. V závislosti na tom, jaký objekt má být vytvořen, je použita jiná komponenta či jejich kombinace.

2.3 Script

Skripty jsou základní složkou všech aplikací, vytvořených v Unity. Většina aplikací potřebuje skripty pro reakci na podněty hráče a k zajištění toho, aby se ve hře odehrávaly události ve správném pořadí. Kromě toho lze skripty použít k vytváření grafických efektů, ovládání fyzikálních vlastností objektů nebo dokonce k implementaci vlastního systému AI pro postavy ve hře. Skripty jsou psány v jazyce C# a každý skript musí dědit ze třídy *MonoBehaviour*.[7]

2.4 Collider

Collider je komponenta, která umožňuje nastavovat tvar objektu instance GameObject za účelem nastavení fyzikálních kolizí.[8] Collider, který je neviditelný, nemusí nutně mít stejný tvar jako těleso, ke kterému se pojí. Pro příklad uvedu právě automobil, kde je zbytečné, aby collider přesně opisoval tvar karoserie vozu. Stačí jen obyčejný kvádr. Důvodem je to, že výpočetní náročnost kolizí stoupá s komplexností tvaru.

2.5 Trigger

Jedná se o Collider, který má nastaven vlastnost **IsTrigger** na true. Triggery slouží pro detekci toho, zdali do oblasti definované jeho tvarem vnikl jiný objekt a zároveň nevytváří žádnou kolizi s tímto vstupujícím objektem. Příkladem může

být finiš, který obsahuje trigger, který když zaznamená, že do něj vjel hráč, vykoná nějakou naprogramovanou událost (ukončí hru, zastaví časomíru, ...).

2.6 Prefab

Prefab je systém, který umožňuje jednoduché znovupoužití herních objektů, se všemi jejich komponentami, nastaveními a podruženými objekty.[9] Z každého herního objektu se dá jednoduše vytvořit prefab. Každá změna na prefabu se pak promítne do všech jeho instancí v herním světe. Takže například máme-li prefab představující strom a rozhodneme se zbarvit jeho listy na modro, pak nám všechny stromy na mapě, vycházející z tohoto stromu, rovněž změní barvu listí na modro.

Zároveň to ale neznamená, že všechny instance prefabu musí mít stejné vlastnosti. Existuje možnost tzv. přetížení nastavení na individuální instanci prefabu. Lze rovněž vytvořit různé varianty prefabu, které umožňují seskupovat množinu přetížených nastavení v lépe použitelné podobě.

3 Tvorba jízdního modelu

Jako první jsem se rozhodl pro vytvoření pojízdného vozidla. Pro začátek jsem použil asset z demo knihovny enginu Unity zvané Standard Assets.[10] Tento asset kromě samotného modelu vozu poskytuje i základní jízdní model, který jsem se rozhodl rozvinout. Problém tohoto modelu byl ten, že nenabízel potřebnou konfigurovatelnost a měl i několik chyb, které způsobovaly zvláštní chování vozu.

Pro úvod zmíním, že naše vozidlo se skládá ze čtyř tzv. Wheel Colliderů, které reprezentují kola vozu. Karoserie vozu je pak reprezentována komponentou Rigidbody. Obě tyto komponenty blíže popíšu v následujících podkapitolách. Auto rovněž obsahuje tři Collider komponenty, které slouží pro detekci kolizí vozu s okolními herními prvky.

3.1 Komponenta Wheel Collider

Jedná se o speciální druh collideru, který je primárně určen pro pozemní vozidla. Má v sobě zabudovaný kolizní detektor, fyzikální model kola a frikční model pneumatik.[11] V tabulce 1 je uveden seznam nejdůležitějších vlastností komponenty Wheel Collider.

Název vlastnosti	Funkce
Mass	Váha vozu.
Radius	Průměr kola.
Suspension Distance	Maximální výška nezatíženého odpružení.
Forward Friction	Hodnota tření pneumatiky v podélném směru.
Sideways Friction	Hodnota tření pneumatiky v příčném směru.

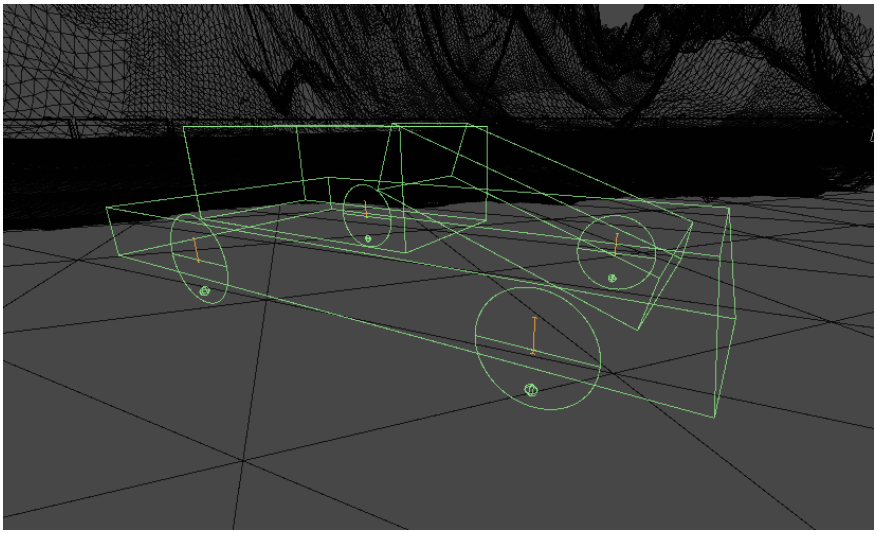
Tabulka 1: Seznam vlastností komponenty Wheel Collider

3.2 Komponenta Rigidbody

Komponenta umožňuje aplikovat fyzikální vlastnosti na herní objekty.[12] Objekty s touto komponentou můžou být vystaveny externím silám a točivému momentu tak, aby se pohybovaly realistickým způsobem. Seznam nejdůležitějších vlastností je uveden v tabulce 2.

Název vlastnosti	Funkce
Mass	Váha objektu.
Drag	Určuje míru zpomalení objektu v důsledku odporu vzduchu. Nulová hodnota znamená nulový odpor vzduchu, nekonečno zastaví objekt okamžitě.
UseGravity	Přepínač určující působení gravitace na objekt.
Velocity	Vektor rychlosti objektu. Představuje rychlost změny polohy.
AngularVelocity	Vektor úhlové rychlosti objektu měřený v radiánech za sekundu.

Tabulka 2: Seznam vlastností komponenty Rigidbody



Obrázek 1: Kostra vozu složená ze základních komponent

3.3 Ovládání vozu

Auto se tedy skládá ze 4 Wheel Colliderů, 3 Box Colliderů a jedné Rigidbody komponenty. Samo o sobě ale toto nestačí k tomu, aby hráč mohl vůz ovládat. K tomu slouží třída *CarUserControl*, konkrétně metoda *FixedUpdate*, která se volá periodicky v pevných časových úsecích (defaultně 0.02 sekundy) a zjišťuje vstup od uživatele. Jelikož Unity umožňuje build pro velké množství platforem, tak není vhodné poslouchat na stisk konkrétních kláves na klávesnici. Lepší je využít metod nabízených třídou *CrossPlatformInputManager*, které tuto akci zobecňují a programátor tak nemusí řešit jejich implementaci pro různá zařízení. Použití lze vidět na ukázce kódu 1.

```
1 private void FixedUpdate()
2 {
3     var horizontalInput = CrossPlatformInputManager.GetAxis("
          Horizontal");
4     var verticalInput = CrossPlatformInputManager.GetAxis("Vertical")
          ;
5     var handbrake = CrossPlatformInputManager.GetAxis("Jump");
6
7     // update pozice vozu
8     m_Car.Move(horizontalInput, verticalInput, verticalInput,
          handbrake);
9 }
```

Zdrojový kód 1: Ukázkový kód zpracování vstupu hráče

Nyní máme zpracovaný vstup od hráče a je na čase tento vstup přetransformovat na pohyb vozu. K tomu slouží třída *CarController*. Tato třída obsahuje public metodu *Move*, která přijímá transformovaný vstup od uživatele a která se stará o zpracování pohybu vozu. V závislosti na vstupních parametrech převádí točivý moment na kola (Wheel Collidery), upravuje úhel zatočení předních kol, aplikuje přítlačnou sílu, vypočítává převodový stupeň a rovněž se stará o audio reprezentaci zvuku motoru, aby odpovídal jeho otáčkám.

3.4 Konfigurovatelnost jízdního modelu

S funkčním jízdním modelem je na čase, aby se dal model upravovat v průběhu hry. K tomu poslouží vypublikování vlastností vozu v rámci třídy *CarController*. Zároveň je ale potřeba toto nastavení ukládat tak, aby bylo dostupné i při dalším spuštění hry. K tomu pomohla třída *PlayerPrefs*, která se stará o ukládání dat hráče. Použití této třídy je vhodné, pokud potřebujeme ukládat data v malém měřítku, funguje totiž na bázi slovníku. Tedy že se hodnoty ukládají pod specifickým klíčem. Komplexní datové typy takto bohužel není možno ukládat.

Ukládání probíhá na třídě *CarTuningMenu*. Jedná se o komponentu (skript) herního objektu reprezentujícího menu s nastavením vozu, které se zobrazí po vjezdu do garáže. Toto menu (obrázek 3) obsahuje seznam posuvníků, kterými lze nastavení vozu měnit. Posuvníky pak vyvolávají public metody, uvedené v ukázce kódu 2.

```
1  public class CarTuningMenu : MonoBehaviour
2  {
3     public CarController carController;
4
5     ...
6
7     // Změna maximální rychlosti
8     public void ChangeTopSpeed(float newSpeed)
9     {
10       carController.TopSpeed = newSpeed;
11       PlayerPrefs.SetFloat(CarTuningConstants.TopSpeed,
             carController.TopSpeed);
12    }
13
14    // Změna přítlačné síly
15    public void ChangeDownforce(float newDownforce)
16    {
17       carController.Downforce = newDownforce;
18       PlayerPrefs.SetFloat(CarTuningConstants.Downforce,
             carController.Downforce);
19    }
20
21    // Změna kontroly trakce
22    public void ChangeTractionControl(float newTractionControl)
23    {
24       carController.TractionControl = Mathf.Round(newTractionControl
             * 10) / 10;
25       PlayerPrefs.SetFloat(CarTuningConstants.TractionControl,
             carController.TractionControl);
26    }
27
28    ...
29
30 }
```

Zdrojový kód 2: Ukázka ukládání vlastností vozu

Pro účely načítání nastavení jsem pak využil metodu *Start* třídy *CarManager*, která se mimo jiné věnuje i restartu pozice vozu při zmáčknutí klávesy „R“.

```
1  public class CarManager : MonoBehaviour
2  {
3     public CarController car;
4
5     // Start se volá před prvním updatem framu
6     void Start()
7     {
8        car.TopSpeed = PlayerPrefs.GetFloat(CarTuningConstants.
             TopSpeed, car.TopSpeed);
9        car.Downforce = PlayerPrefs.GetFloat(CarTuningConstants.
             Downforce, car.Downforce);
10       car.TractionControl = PlayerPrefs.GetFloat(CarTuningConstants.
             TractionControl, car.TractionControl);
11       car.Power = PlayerPrefs.GetInt(CarTuningConstants.Power, car.
             Power);
12       car.Height = PlayerPrefs.GetFloat(CarTuningConstants.Height,
             car.Height);
13       car.DriveType = (CarDriveType)PlayerPrefs.GetInt(
             CarTuningConstants.DriveType, (int)car.DriveType);
14       car.BrakesEfficiency = PlayerPrefs.GetFloat(CarTuningConstants
             .BrakesEfficiency, car.BrakesEfficiency);
15    }
16 }
```

Zdrojový kód 3: Ukázka načítání vlastností vozu

Při pohledu na ukázkové kódy si možná čtenář klade otázku, proč jsem načítání/ukládání nastavení neaplikoval až v *setterech*[2], respektive *getterech*[3]. Bylo to z toho důvodu, že jsem chtěl oddělit funkčnost správy nastavení od jízdního modelu. Navíc by nebylo zcela vhodné, aby se při čtení vlastností muselo znovu a znovu vyhledávat odpovídající uživatelské nastavení. To by při budoucím rozšíření jízdního modelu mohlo způsobovat výkonnostní problémy.

[2]Jako setter se v informatice označuje funkce, která je volaná před uložením nové hodnoty vlastnosti do její privátní proměnné.

[3]Analogicky k setteru, jedná se o funkci, která je volaná při získávání hodnoty vlastnosti.

4 Tvorba herních mechanik

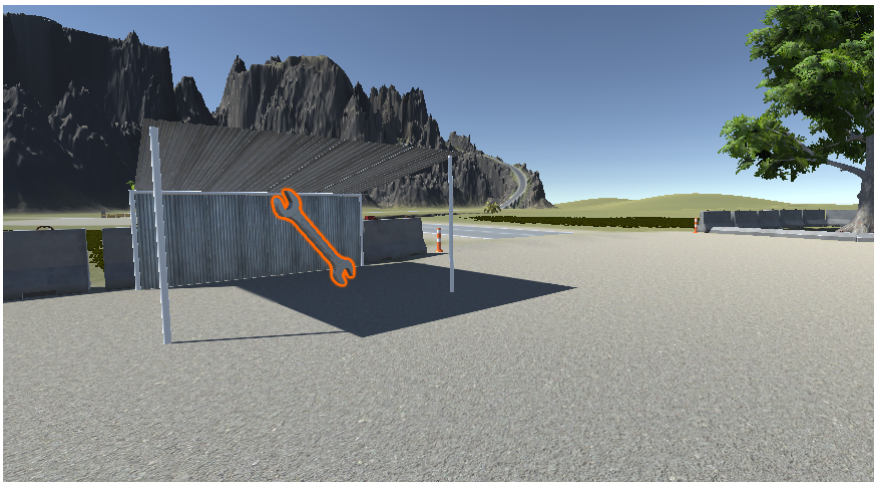
S vyvinutým jízdním modelem bylo na čase začít přemýšlet, jak jej v zábavné formě implementovat do hry. Jak jsem již v úvodní kapitole 1 zmínil, mojí inspirací byly herní série TrackMania a SpinTires. Rozhodl jsem se proto jejich prvky zkombinovat a vytvořit hru, jejíž cílem bude v co nejkratším časovém úseku zdolat vrcholek hory a dojet do cíle. Zároveň ale k cíli nevede pouze jedna cesta. To bude nutit hráče k opakovanému hraní a nalezení časově nejkratší cesty. V následujících podkapitolách podrobně popíši základní herní prvky, se kterými se hráč setká.

4.1 Startovací oblast

Startovací oblast je místo, ve kterém se hráč poprvé objeví po spuštění hry. Při spuštění hry se hráči rovněž zobrazí odpočet, který odstartuje jeho cestu na vrchol hory spolu s časomírou.

4.1.1 Garáž

Gáráž umožňuje hráči, aby si upravoval vlastnosti vozu. V závislosti na tom, zda hráč novým nastavením vlastnosti vozu zhorší/zlepší, se mu přičtou/odečtou vteřiny do tzv. Tuning bonusu, respektive Tuning penalty v pravé horní části obrazovky.

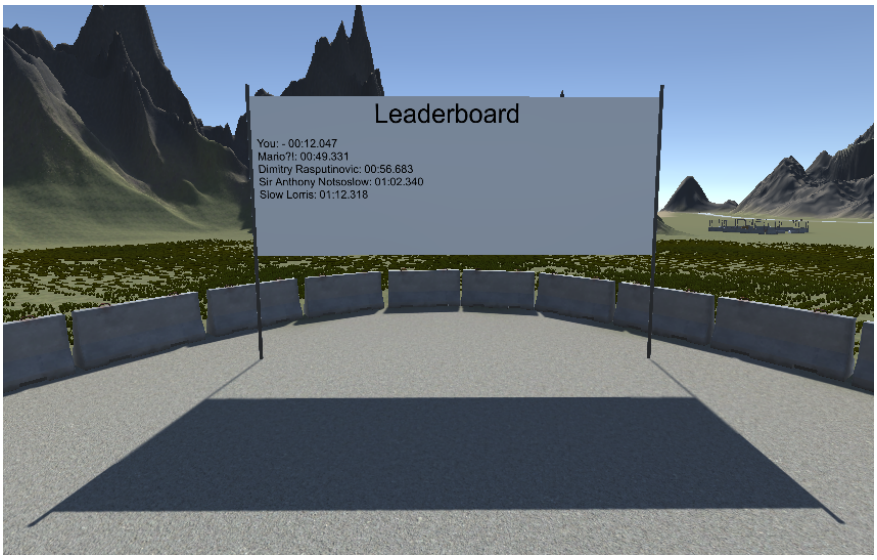


Obrázek 2: Garáž

Obrázek 3: Ukázka menu s nastavením vozu (nefinální verze)

4.1.2 Tabulka výsledků

Na tabulce výsledků, která je vedena ve formě billboardu, se zapisují nejlepší časy. Tabulka obsahuje seznam předdefinovaných časů, a hráčův výsledek se do ní zapíše až po prvním úspěšném dokončení závodu. Hráčův čas se ukládá spolu s nastavením vozu a jedná se o perzistentní data, která se načtou i při příštím spuštění hry.



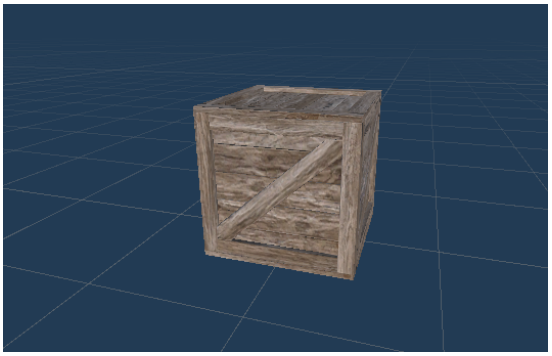
Obrázek 4: Tabulka výsledků

4.2 Časové bedny

Po mapě jsou rozmístěny tzv. časové bedny. Těchto beden je několik druhů a každá z nich má po jejím sražení jiný efekt na časomíru. Všechny druhy beden vycházejí z jednoho modelu, kterému jsem jen pozměnil zbarvení textury, případně jeho velikost.

4.2.1 Time crate

První typ bedny způsobuje, že se časomíra zbarví do žluta a zastaví se na 3 vteřiny. Tento efekt je kumulativní, takže pokud hráč srazí najednou 2 bedny, zastaví se čas na 6 vteřin. Naprogramování této funkčnosti bylo docela zajímavé a vyžadovalo využití synchronizačních primitiv. Hráč totiž může srazit vícero beden najednou a bylo potřeba vědět, jak dlouho se má vykonávat koprogram (v Unity pod názvem Coroutine) zastavující čas. K tomu posloužila statická proměnná *_numberOfCoroutines*, která je inkrementována pokaždé, když hráč vrazí do bedny a dekrementována poté, co doběhne daný koprogram, který zastavil časomíru na 3 vteřiny. Vzhledem k tomu, že ale inkrementace probíhá asynchronně, musel jsem využít metod třídy Interlocked, které se starají o synchronizovanou inkrementaci/dekrementaci číselné hodnoty poskytnuté přes referenci. Zmiňovanou část kódu lze vidět na ukázce 4.



Obrázek 5: Time crate

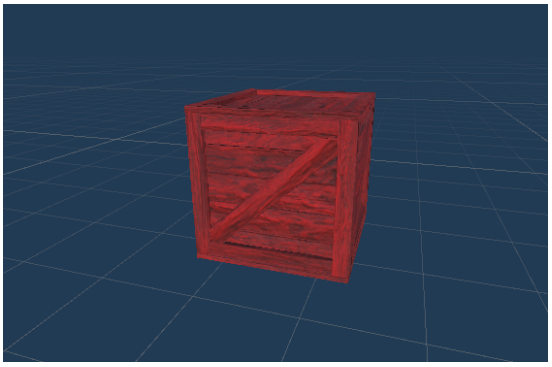
```
1  private static int _numberOfCoroutines;
2
3  ...
4
5  // Tato metoda se spouští tehdy, když hráč narazí do bedny
6  private void OnTriggerEnter(Collider other)
7  {
8
9     ...
10
11    Interlocked.Increment(ref _numberOfCoroutines);
12
13    if (_numberOfCoroutines == 1)
14       StartCoroutine(Wait3Seconds());
15 }
16
17 private static IEnumerator Wait3Seconds()
18 {
19    TrialTimeManager.IsTimeCrateEffectActive = true;
20
21    while (_numberOfCoroutines != 0)
22    {
23       TrialTimeManager.timer.Stop();
24       yield return new WaitForSeconds(3);
25       Interlocked.Decrement(ref _numberOfCoroutines);
26    }
27
28    TrialTimeManager.IsTimeCrateEffectActive = false;
29
30    if (!TrialTimeManager.IsGameFinished)
31       TrialTimeManager.timer.Start();
32 }
```

Zdrojový kód 4: Implementace efektu zastavení času

4.2.2 Evil crate

Tento typ bedny je zbarven do červena a způsobuje, že se hráči do časomíry přičtou 2 sekundy.

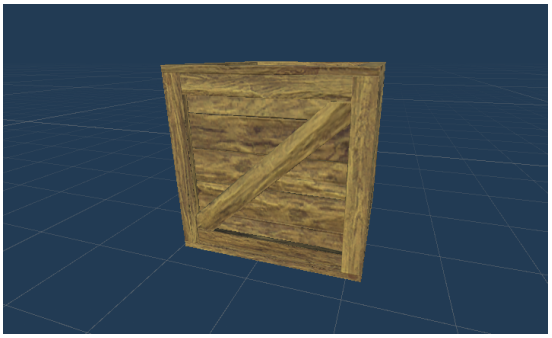
Implementace efektu této bedny se z počátku zdála velice přímočará, stačí k aktuálnímu času časomíry, který je reprezentován pomocí třídy Stopwatch, přičíst 2 sekundy. Bohužel, tato třída neobsahuje jednoduchý způsob, jak inkrementovat hodnotu časomíry. Byl jsem nakonec nucen si vytvořit vlastní implementaci nazvanou *CustomStopwatch* 5. V podstatě se jedná o návrhový vzor dekorátor, kdy dědím ze třídy *Stopwatch* a rozšiřuji její možnosti o přidávání libovolného časového úseku.



Obrázek 6: Evil crate

4.2.3 Mega crate

Efekt posledního druhu bedny je ten, že způsobuje vyresetování časomíry. Tento druh bedny je velmi vzácný a na mapě se moc nevyskytuje. Bedny jsou větší než předchozí dva typy a jsou zbarveny do žluta.

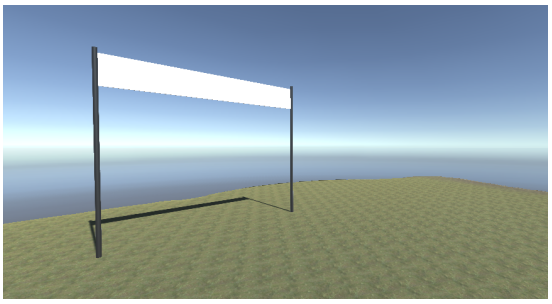


Obrázek 7: Mega crate

Při tvorbě bylo nutno věnovat zvýšenou pozornost tomu, zdali v době sražení bedny byla časomíra spuštěna či pozastavena. V případě, že byla spuštěna, volala se metoda *Restart*. V opačném případě se volala metoda *Reset*. Metody se liší v tom, že zatímco *Restart* časomíru vynuluje a opět spustí, *Reset* časomíru pouze vynuluje. Tyto metody lze vidět ve zdrojovém kódu třídy *CustomStopwatch* 5, kde jsem musel vytvořit přetížení jejich defaultní funkčnosti z toho důvodu, že jsem zároveň potřeboval vynulovat proměnnou reprezentující časovou penaltu, která se přičítá k aktuálnímu stavu časomíry.

4.3 Finiš

Jedná se o klasický finiš v podobě, jaké ho známe i z reálných závodů. Když jím hráč projede, časomíra se zastaví, hráčův čas se uloží a je zobrazen na tabulce výsledků ve startovací oblasti, kam je hráč vyzván k návratu pomocí stisku klávesy „R“.



Obrázek 8: Finiš

```
1  public class CustomStopwatch : Stopwatch
2  {
3      private TimeSpan _penalty = TimeSpan.Zero;
4
5      public new TimeSpan Elapsed => base.Elapsed + _penalty;
6
7      public void AddPentaly(TimeSpan penaltyDuration)
8      {
9          _penalty += penaltyDuration;
10     }
11
12     public new void Reset()
13     {
14         base.Reset();
15         _penalty = TimeSpan.Zero;
16     }
17
18     public new void Restart()
19     {
20         base.Restart();
21         _penalty = TimeSpan.Zero;
22     }
23 }
```

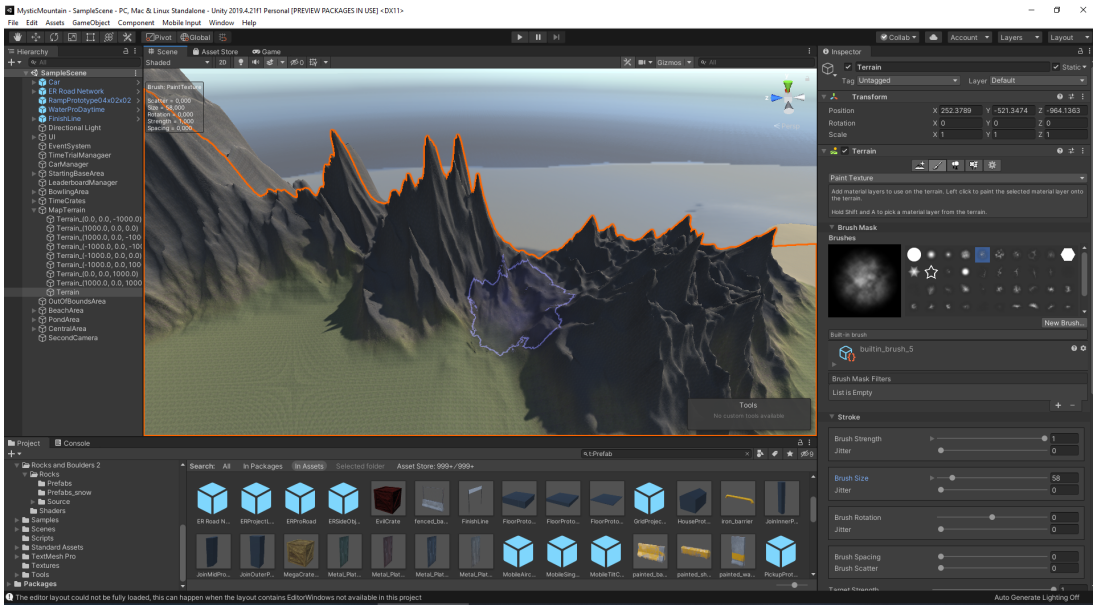
Zdrojový kód 5: Rozšíření třídy Stopwatch

5 Tvorba herní mapy a uživatelského rozhraní

Nyní, když jsou herní mechaniky dokončeny, je na čase vytvořit herní mapu/terén. Po důkladném rozmýšlení jsem se rozhodl terén nastylovat jako ostrov v subtropickém podnebném pásu. Ostrov jsem vybral z toho důvodu, že nejméně očividným způsobem maskuje hranice herní mapy. Zároveň implementovat nekonečný horizont v podobě moře je snadnější, než například olemovat mapu nepřekonatelným pohořím, či hůř, neviditelnými zdmi[4].

5.1 Terén

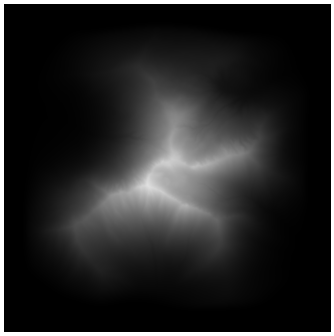
Engine Unity nabízí řadu nástrojů usnadňujících tvorbu terénu. Vývojář začíná tím, že vytvoří rovinný terén, kterému nadefinuje rozměry, případně další parametry. Terén se pak tvaruje pomocí tzv. štětců, které podle nastavené intenzity a zvolené heightmapy[5] deformují povrch terénu. Díky těmto nástrojům se mi jednoduše povedlo vytvořit celou mapu ostrova, který na sobě má pohoří, kam má hráč za úkol se dostat. Příklad heightmapy lze vidět na obrázku 10.



Obrázek 9: Ukázka práce se štětcem

[4]Neviditelnými zdmi se rozumí hranice, kterou hráč není schopen překročit, ale není žádný vizuální podnět, který by mu indikoval, že se jedná o konec mapy. [13]

[5]Heightmap je rastrový obrázek, který reprezentuje rozdíly výšek na ploše pomocí odstínů šedé. Typicky bývá určena černá barva pro nejnižší místa a bíla pro nejvyšší. [14]



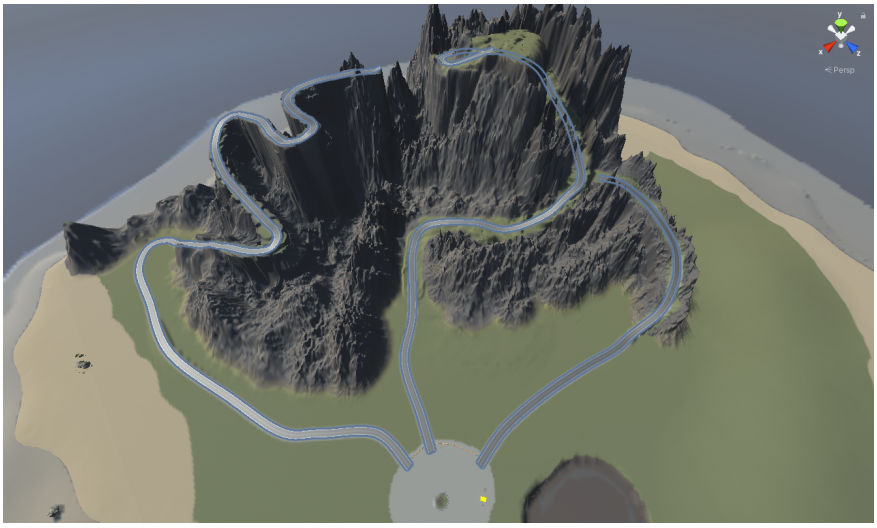
Obrázek 10: Příklad heightmapy

Ostrov je obklopen mořem, a tak přirozeně přišla myšlenka, co se stane, když hráč do moře vjede. Nakonec jsem se inspiroval videoherní sérií Tony Hawk's[15], kde při podobné události je hráčova pozice vyresetována a na obrazovku je hráči vypsána hláška, v čem udělal chybu. Nutno podotknout, že tyto hlášky bývají často humorné. Implementace byla provedena pomocí rozlehlého triggeru umístěného kousek pod vodní hladinu, který při detekci toho, že do něj hráč vjel, vykoná výše zmíněné chování.

5.2 Cesty

Pro tvorbu cest jsem se rozhodl využít rozšíření EasyRoads3D ve verzi pro použití zdarma.[16] Toto rozšíření umožňuje jednoduše vytvářet síť cest, přidávat objekty v okolí cesty (svodidla, dopravní cedule, ...) a hlavně editovat terén tak, aby do něj silnice správně zapadala.

Co se herního hlediska týče, rozhodl jsem se pro 3 možné cesty vedoucí na vrchol hory. Každá vyžaduje od hráče jinou strategii a je pro ni vhodné jiné nastavení vozidla.



Obrázek 11: Pohled na větvící se cesty

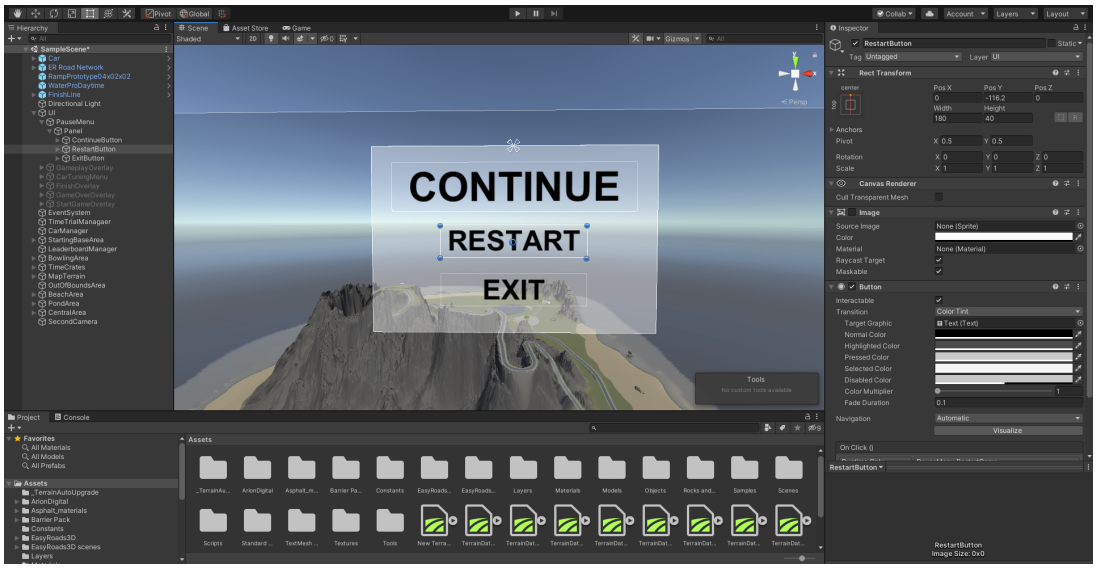
5.3 Uživatelské rozhraní

Vývoj samotného uživatelského rozhraní není nijak náročný a kdo již někdy v minulosti dělal s technologiemi WPF[6] či WinForms[7], tak se bude cítit jako doma. Z technické stránky se jedná opět o herní objekt, který implementuje komponentu Canvas (dále jen „plátno“). Do tohoto plátna pak vývojář může libovolně vkládat text či jiné základní prvky UI (např. slider, toggle, scrollbar, . . .). Těmto prvkům se pak dají velice jednoduše pomocí C# skriptů nastavovat nejenom jejich hodnoty, ale i vzhled či pozice. Ve své práci jsem vytvořil 6 různých pláten, u kterých pouze přepínám jejich viditelnost na základě situace ve hře.

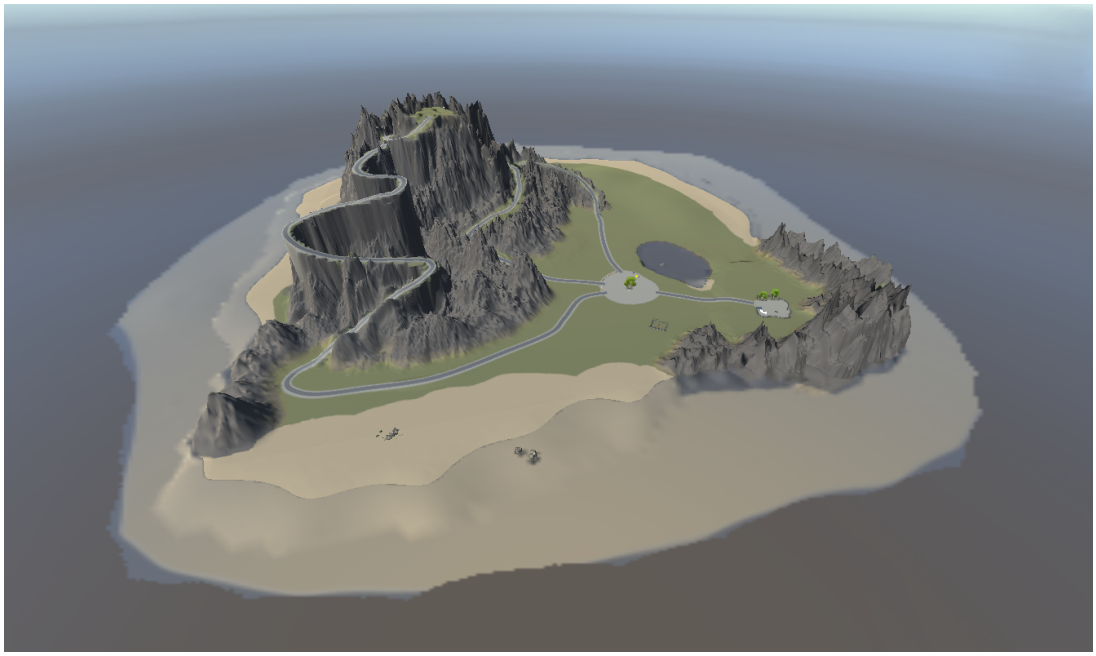
1. PauseMenu – menu zobrazené při pozastavení hry
2. GameplayOverlay – plátno zabrazující časomíru a rychlost
3. CarTuningMenu – menu pro úpravu auta, zobrazené při vjezdu do garáže
4. FinishOverlay – zobrazuje nápis „Finish“ při průjezdu cílem
5. GameOverOverlay – zobrazuje text s odůvodnění hráčova „úmrtí“
6. StartGameOverlay – plátno úvodní obrazovky hry

Všechna tato plátna by se samozřejmě dala vložit pouze do jednoho, ale vyžadovalo by to zbytečně složitou logiku na naprogramování, která by se obtížně ladila. Jelikož logika tvorby je pro každé plátno stejná, uvedu detailní popis pouze u jednoho z nich, konkrétně u plátna nazvaného PauseMenu. Jedná se o klasické menu zobrazené při pozastavení hry, tak jak ho můžeme znát z jiných her. Menu umožňuje kromě opětovného pokračování ve hře ještě restartování úrovně, případně ukončení celé aplikace a návrat do systému. Plátno je řízeno pomocí skriptu *PauseMenu.cs*, který obsahuje metodu *Update* starající se o registraci stisknutí klávesy escape a následné zobrazení/skrytí menu. Dále tento skript obsahuje obslužné události, které se vykonají při kliknutí na jejich přiřazení tlačítko. Samotné menu můžete vidět na obrázku 12.

[6] Windows Presentation Foundation [17]
[7] Windows Forms [18]



Obrázek 12: Ukázka práce na UI prvku PauseMenu



Obrázek 13: Pohled na finální ostrov z ptačí perspektivy

Závěr

Cílem práce bylo vytvořit konfigurovatelný jízdní model vozidla a implementovat jej do závodní videohry. Oba tyto cíle jsem splnil a výsledkem je hra zvaná Mystic Mountain, kde má hráč možnost upravovat si vlastnosti vozu, které mají vliv na jeho ovladatelnost. Hráč je ale zároveň herními mechanikami nucen zvážit, jaké nastavení vozu zvolí.

Díky této práci jsem si osvojil tvorbu her s enginem Unity. Zároveň jsem mohl nahlédnout do světa videoherního vývoje a porovnat jej s mou dosavadní praxí vývojáře webových aplikací. Oba světy si jsou velice podobné. Herní vývoj mi přijde svobodnější a tím i zábavnější. Na druhé straně ale nemusí být z pracovního hlediska tak stabilní, jako vývoj webových aplikací.

Aplikaci bych chtěl do budoucna dále rozvíjet a třeba i jednou vydat jako hru, kterou si bude moci zahrát široká veřejnost.

Conclusions

The goal of the thesis was to create a configurable driving model of a vehicle and implement it into a racing video game. I met both goals and the result is a game called Mystic Mountain, where the player has the opportunity to adjust the properties of the car, which affect its handling. However, the player is also forced by game mechanics to consider which car settings to choose.

Thanks to this thesis, I learned to create games with Unity engine. At the same time, I was able to investigate the world of video game development and compare it with my previous practice as a web application developer. Both worlds are very similar. Game development strikes me as freer and thus more fun. On the other hand, it may not be as stable job as web application development.

I would like to further develop the application in the future and even release it once as a game that the general public will be able to play.

A Obsah přiloženého CD/DVD

Přiložené DVD obsahuje 3 složky a jeden textový soubor.

bin/
: Složka obsahuje všechny soubory potřebné ke spuštění hry. Samotná hra se spouští pomocí souboru *MysticMountain.exe*.

doc/
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

src/
: Kompletní zdrojové kódy a soubory pro otevření hry v enginu Unity.

readme.txt
: Soubor s instrukcemi pro spuštění hry a základním návodem k ovládání.

U veškerých cizích převzatých materiálů obsažených na CD/DVD jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro všechny použité (a citované) materiály, u kterých toto není splněno a nejsou tak obsaženy na CD/DVD, je uveden jejich zdroj (např. webová adresa) v bibliografii nebo textu práce nebo v souboru `readme.txt`.

Literatura

[1] UBISOFT NADEO. *Trackmania* [online]. [cit. 2021-4-20]. Herní série. Dostupný z: ⟨https://www.trackmania.com⟩.

[2] OOVEE GAMES. *SpinTires* [online]. [cit. 2021-4-20]. Herní série. Dostupný z: ⟨https://www.epicgames.com/store/en-US/p/snowrunner⟩.

[3] UNITY TECHNOLOGIES. *Plans and pricing* [online]. [cit. 2021-4-20]. Dostupný z: ⟨https://store.unity.com/⟩.

[4] UNITY TECHNOLOGIES. *Licensing & activation* [online]. [cit. 2021-4-20]. Dostupný z: ⟨https://unity3d.com/unity/faq/2491⟩.

[5] UNITY MANUAL. *Scenes - Unity Manual* [online]. [cit. 2021-4-20]. Dostupný z: ⟨https://docs.unity3d.com/Manual/CreatingScenes.html⟩.

[6] UNITY MANUAL. *GameObject - Unity Manuals* [online]. [cit. 2021-4-20]. Dostupný z: ⟨https://docs.unity3d.com/Manual/GameObjects.html⟩.

[7] UNITY MANUAL. *Scripting - Unity Manuals* [online]. [cit. 2021-4-20]. Dostupný z: ⟨https://docs.unity3d.com/Manual/ScriptingSection.html⟩.

[8] UNITY MANUAL. *Colliders - Unity Manual* [online]. [cit. 2021-4-20]. Dostupný z: ⟨https://docs.unity3d.com/Manual/CollidersOverview.html⟩.

[9] UNITY MANUAL. *Prefabs - Unity Manual* [online]. [cit. 2021-4-20]. Dostupný z: ⟨https://docs.unity3d.com/Manual/Prefabs.html⟩.

[10] UNITY TECHNOLOGIES. *Standard Assets - Unity Asset Pack* [online]. [cit. 2021-4-20]. Dostupný z: ⟨https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-for-unity-2018-4-32351⟩.

[11] UNITY MANUAL. *Wheel Collider - Unity Manual* [online]. [cit. 2021-4-20]. Dostupný z: ⟨https://docs.unity3d.com/Manual/class-WheelCollider.html⟩.

[12] UNITY MANUAL. *Rigidbody - Unity Manual* [online]. [cit. 2021-4-20]. Dostupný z: ⟨https://docs.unity3d.com/Manual/class-Rigidbody.html⟩.

[13] INVISIBLE WALL. *Invisible wall: Wikipedia, The Free Encyclopedia* [online]. [cit. 2021-4-20]. Dostupný z: ⟨https://en.wikipedia.org/wiki/Invisible_wall⟩.

[14] HEIGHTMAP. *Heightmap: Wikipedia, The Free Encyclopedia* [online]. [cit. 2021-4-20]. Dostupný z: ⟨https://en.wikipedia.org/wiki/Heightmap⟩.

[15] VICARIOUS VISIONS. *Tony Hawk's Pro Skater* [online]. [cit. 2021-4-20]. Dostupný z: ⟨https://www.tonyhawkthegame.com/⟩.

[16] ANDASOFT. *EasyRoads3D Free - Unity Asset Pack* [online]. [cit. 2021-4-20]. Dostupný z: ⟨https://assetstore.unity.com/packages/3d/characters/easyroads3d-free-v3-987⟩.

[17] MICROSOFT DOCS. *Microsoft Docs - WPF overview* [online]. [cit. 2021-4-20]. Dostupný z: ⟨https://docs.microsoft.com/dotnet/desktop/wpf/introduction-to-wpf?view=netframeworkdesktop-4.8⟩.

[18] MICROSOFT DOCS. *Windows Forms overview* [online]. [cit. 2021-4-20]. Dostupný z: ⟨https://docs.microsoft.com/dotnet/desktop/winforms/windows-forms-overview?view=netframeworkdesktop-4.8⟩.