C# - 1 - Aproximace Ludolfova čísla π

Pracovní úkoly:

1. Vytvořte celkem 2 funkce s jediným vstupním parametrem `N`, jejichž výstupem bude aproximace čísla π určená pomocí **součtu** prvních `N` prvků následujících posloupností:
   - Eulerova: $a_1 = 1, a_n = \frac{1}{n^2} \Rightarrow \frac{\pi^2}{6} = \sum_{n=1}^{\infty} a_n$.
   - Leibnizova: $a_1 = 1, a_n = \frac{(-1)^{n-1}}{2n-1} \Rightarrow \frac{\pi}{4} = \sum_{n=1}^{\infty} a_n$.
2. Vytvořte funkci s jediným vstupním parametrem `N`, jejímž výstupem bude aproximace čísla π určená **součinem** prvních `N` členů následující **rekurentní** posloupnosti:
   - Viétova: $a_1 = \sqrt{\frac{1}{2}}, a_n = \sqrt{\frac{1}{2} + \frac{1}{2} a_{n-1}} \Rightarrow \frac{2}{\pi} = \prod_{n=1}^{\infty} a_n$.
3. Vytvořte funkci, která seřadí tyto tři výše zmíněné aproximace čísla π podle rychlosti jejich konvergence pro danou požadovanou přesnost. Jedná se o matematickou konvergenci, takže nejrychleji konverguje ta aproximace, která dosáhne požadované přesnosti pro nejmenší `N`.

C# - 2 - Výpočet obsahu plochy pod grafem funkce

Pomocný text: *Integrál*

Pracovní úkoly:

1. Vytvořte funkci `integral(f, a, b, n)`:
   - Vstupy:
     - `f` … funkce (jedná se o klasickou funkci v C#).
     - `a` … reálné číslo (menší než `b`).
     - `b` … reálné číslo (větší než `a`).
     - `n` … počet dělení intervalu (`a`, `b`).
   - Výstup: obsah plochy pod grafem funkce `f` na intervalu (`a`, `b`), který vznikne aproximací této plochy pomocí `n` obdélníků.
2. Vytvořte funkci, která porovná výstup funkce `integral()` s přesným matematickým výsledkem pro obecné lineární funkce `f`.

Alternativní postup:

Vycházíte z již existujícího C# projektu, který je možné získat z následujícího GitHub repozitáře: https://github.com/JindrichDvorak/BasicGraphicsEngine-Project.

Pracovní úkoly:

1. Pomocí dvou objektů třídy `Line`, nebo `Quad` vykreslete dvě vzájemně kolmé „přímky“ reprezentující souřadnicové osy x, y.
2. Vytvořte metodu `DrawFunc(f, a, b, n)`, která pomocí `n` objektů třídy `Particle` vykreslí graf funkce `f` na intervalu (`a`, `b`).
3. Vytvořte novou třídu `Bar`, která dědí z třídy `Line`[1]. Konstrukce objektu třídy `Bar` by měla vyžadovat zadání počátečního bodu, šířky, výšky a barvy tak, aby se vykreslil obdélník daných rozměrů a pro jeho podstavu (úsečka s délkou zadané šířky) platilo, že je rovnoběžná s osou x a její střed byl shodný s počátečním bodem.
4. Modifikujte metodu `DrawFunc()` tak, aby kromě vykreslení grafu funkce `f` i vyplnila plochu pod křivkou tohoto grafu pomocí `n` objektů třídy `Bar`.
5. Třídu `Bar` rozšiřte o funkci `GetArea()`, která vrací obsah obdélníku, který třída `Bar` vykresluje. Následně pomocí této funkce modifikujte metodu `DrawFunc()` tak, aby do konzole vypsala obsah vykreslené plochy pod křivkou.

[1] Třída `Line` vytváří obdélník, který je definován šířkou, počátečním a koncovým bodem. Nejprve vznikne úsečka, kterou určuje počáteční a koncový bod a následně je tato úsečka „rozšířena“ na obdélník dané šířky tak, aby původní úsečka byla osou symetrie tohoto obdélníku pro jeho výšku.

C# - 3 - Vlastní datový typ: vektor

Pomocný text: *Vektory*

Pracovní úkoly:

1. Vytvořte třídu implementující 2D vektor s následujícími vlastnostmi:
   - Operátor (binární) „+“ přetížen na sčítání vektorů.
   - Operátor (binární) „−“ přetížen na odčítání vektorů.
   - Operátor (unární) „−“ přetížen tak, aby při použití na vektor vrátil jeho opačný vektor.
   - Operátor (binární) „*“ přetížen na součin skaláru a vektoru (v tomto pořadí).
   - Operátor (binární) „*“ přetížen na skalární součin vektorů.
   - Funkce `len()`, která vrací délku tohoto vektoru.
   - Funkce `dev(v)`, která vrací odchylku tohoto vektoru a zadaného vektoru `v`.
   - Funkce `dir()`, která vrací vektor jednotkové délky ve směru tohoto vektoru.
   - Funkce `normal(right)` s nepovinným parametrem `right`, která vrací vektor jednotkové délky, který je kolmý na tento vektor (normálový vektor). Pokud má parametr `right` hodnotu `true` (defaultní hodnota), tak tento vektor a jeho normálový vektor tvoří pravotočivou soustavu (normálový vektor je vůči tomuto vektoru otočený v protisměru hodinových ručiček), v opačném případě tvoří levotočivou soustavu.
   - Funkce `rotate(angle)`, která vrací vektor, jenž vzniknul otočením tohoto vektoru o úhel `angle` ve stupních (pozor na správné úhlové jednotky).
   - Metodu `print()`, která přehledně vypíše tento vektor do konzole.
   - Statické varianty všech předchozích funkcí a metod (ne operátorů).
2. Pomocí této třídy vytvořte metody, které řeší následující úlohy a jejich formátované výsledky vypíší do konzole. **Minimálně jednu z následujících úloh vyřešte pouze pomocí operátorů, statických funkcí a statických metod**: Uvažujte tři nekolineární body $A = [0,\ 0]$, $B = [5,\ 0]$ a $C = [3,\ 5]$, vypočtěte:
   - Délku úsečky AB.
   - Střed úsečky AB.
   - Zdali je trojúhelník ABC pravoúhlý.
   - Obsah trojúhelníku ABC.
   - Souřadnice bodů A', B' a C', které vznikly otočením trojúhelníku ABC kolem bodu A o úhel 90°.

Zajímavé rozšíření:

Přesuňte své řešení do již existujícího C# projektu, který je možné získat z následujícího GitHub repozitáře: https://github.com/JindrichDvorak/BasicGraphicsEngine-Project.

- Vytvořte si souřadné osy pomocí dvou objektů `Quad`.
- Vizualizujte výsledky všech úloh z pracovního úkolu 2 pomocí objektů `Line` (samotné body A, B a C můžete vykreslit pomocí objektů `Particle`).[2]

[2] Bude potřeba vytvořit funkci, která vámi implementovaný vektor konvertuje na vektor typu `Vector2`.

C# - 4 - Vlastní datový typ: matice

Pomocný text: *Matice*

Pracovní úkoly:

1. Vytvořte třídu implementující matici typu $m \times n$ s následujícími vlastnostmi:
   - Operátor (binární) „+“ přetížen na sčítání matic.
   - Operátor (binární) „−“ přetížen na odčítání matic.
   - Operátor (unární) „−“ přetížen tak, aby při použití na matici vrátil matici obsahující opačné prvky.
   - Operátor (binární) „*“ přetížen na součin skaláru a matice (v tomto pořadí).
   - Operátor (binární) „*“ přetížen na součin dvou matic.
   - Operátor (binární) „*“ přetížen na součin matice a vektoru (v tomto pořadí). Vektor implementujte jako matici o jediném sloupci -- tohoto omezení na jediný sloupec, nebo řádek (viz následující bod), dosáhnete pomocí nové třídy, která bude dědit z třídy matice, v tomto zadání ji označíme jako `LineMatrix`.
   - Operátor (binární) „*“ přetížen na součin vektoru a matice (v tomto pořadí). Vektor implementujte jako matici o jediném řádku pomocí `LineMatrix`.
   - Funkce `trace()`, která vrací součet všech prvků na diagonále této matice.
   - Funkce `trans()`, která vrací matici, jejíž prvky vznikly prohozením řádků a sloupců této matice – takzvaná transpozice.
   - Metoda `print()`, která přehledně vypíše tuto matici do konzole.
   - Statické varianty všech předchozích funkcí a metod (ne operátorů).
2. Pomocí této třídy vytvořte metody, které řeší následující úlohy a jejich formátované výsledky vypíší do konzole:
   - Definujte a vypište následující matice (jako objekty třídy `Matrix`): $R = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$, $T = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$, $C = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. Dále definujte a vypište vektory (jako objekty třídy `LineMatrix`): $\vec{v} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ a $\vec{w} = (1,\ 1)$.
   - Součet matic: $R + T$.
   - Rozdíl matic pomocí: $R - T$ a pomocí: $R + (-T)$.
   - Součin matic: $A = RT$.
   - Součin matic: RC.
   - Součin matice a vektoru: $A\vec{v}$.
   - Transpozice matice C.
   - „Trace“ matice T.

Zajímavé rozšíření:

Přesuňte své řešení do již existujícího C# projektu, který je možné získat z následujícího GitHub repozitáře: https://github.com/JindrichDvorak/BasicGraphicsEngine-Project.

- Vytvořte si souřadné osy pomocí dvou objektů `Quad`.
- Vizualizujte všechny vektory z pracovního úkolu 2 pomocí objektů `Line`. Jaký význam má matice R a T? Jak by vypadala matice, která „prodlužuje“ vektory? Existuje obecný tvar matice R?

C# - 5 - Detektor kolizí

Pomocný text: *Kolize základních 2D objektů*

Vycházíte z již existujícího C# projektu, který je možné získat z následujícího GitHub repozitáře: https://github.com/JindrichDvorak/BasicGraphicsEngine-Project.

Pracovní úkoly:

1. Vytvořte následující dvojice různě velkých objektů daných typů:[3]
   - Nepřekrývající se kruhy.
   - Překrývající se kruhy.
   - Nepřekrývající se obdélníky.
   - Překrývající se obdélníky.
   - Nepřekrývající se kruh a obdélník.
   - Překrývající se kruh a obdélník.
2. Vytvořte **statickou** třídu `CollisionDetector`, pomocí které bude možné detekovat, zdali se dané dva objekty překrývají. Základními funkcemi této třídy budou tedy (příklady možných názvů):[4]
   - `CircleCircle(circle1, circle2)` – detekce kolize dvou kruhů.
   - `QuadQuad(quad1, quad2)` – detekce kolize dvou obdélníků.
   - `CircleQuad(circle, quad)` – detekce kolize kruhu a obdélníku.
3. Pomocí těchto funkcí manuálně ověřte, zdali došlo ke kolizi mezi relevantními objekty z pracovního úkolu 1. Pokud ke kolizi došlo, tak tento výsledek kreativně vizualizujte (například, že objektům změníte barvu).
4. Ve statické třídě vytvořte metodu `DetectCollisions(objects)`, která přijme všechny objekty ve vaší scéně a určí, jaké dvojice objektů se překrývají (například, že překrývajícím objektům změní barvu) – takto prakticky automatizujete pracovní úkol 3.

[3] Doporučujeme, aby každý objekt v dané dvojici měl jinou barvu s alfa kanálem menším než 1, dále doporučujeme vždy u jednoho objektu z dané dvojice nastavit odlišnou souřadnici z (pak v kombinaci s předchozím doporučením bude možné lépe vidět překryté části objektů).

[4] Geometrický střed každého objektu je možné získat member funkcí `GetPosition2D()`, která vrací dvourozměrný vektor `Vector2` obsahující souřadnice x a y (odlišné souřadnice z různých objektů tedy nijak neovlivní detekci jejich „2D kolize“).

C# - 6 - Databáze v C#

Vycházíte z již existujícího C# projektu, který je možné získat z následujícího GitHub repozitáře: https://github.com/JindrichDvorak/SQLiteCs-Project.git.

Pracovní úkoly:

1. Vytvořte metodu `PrintDataTable()`, která přijme výstup funkce `Query()`, tedy object typu `QueryResult` a přehledně jej vykreslí do konzole jako tabulku (nemusí být ohraničená).
2. Vytvořte metodu `PrintQueryResult()`, která bude mít stejný výstup jako `PrintDataTable()`, ale místo objektu typu `QueryResult` přijímá SQL příkaz ve formě stringu.
3. V již existující databázi `csfd.db` (jediná tabulka obsahující 100 nejlépe hodnocených filmů podle ČSFD) určete čistě pomocí SQL „příkazů“ (vypsáním výsledků do konzole):
   - Kolik filmů v databázi má český původ?
   - Nejlépe hodnocený český film v této databázi.
   - Nejhůře hodnocený český film v této databázi.
   - Názvy a roky vydání všech filmů v této databázi, ve kterých hraje herec Christian Bale.
4. Založte novou databázi, ve které následně pomocí SQL „příkazů“ vytvoříte dvě navzájem propojené tabulky (název tabulky: 1. sloupec; … ; N-tý sloupec):
   - Autor: id; jméno; rok narození.
   - Kniha: id; název; id autora; rok vydání; žánr.
5. Do nově vzniklé databáze vložte alespoň tři autory, přičemž každému autorovi přidejte alespoň dvě knihy (můžete se inspirovat seznamem níže).
6. V této databázi následně určete čistě pomocí SQL „příkazů“ (vypsáním výsledků do konzole):
   - Tabulku, která bude obsahovat veškerá data z obou tabulek.
   - Tabulku ve tvaru (sloupce): jméno autora; počet knih náležících danému autorovi v knihovně.
   - Tabulku ve tvaru (sloupce): jméno autora; rok vydání jeho nejstaršího díla; rok vydání jeho nejnovějšího díla.

Seznam autorů pro inspiraci:

J. R. R. Tolkien (1892) -- fantasy	
The Hobbit (1937)	LotR: The Fellowship of the Ring (1954)
LotR: The Two Towers (1954)	LotR: The Return of the King (1955)
Daniel Abraham (1969) – sci-fi	
Leviathan Wakes (2011)	Caliban’s War (2012)
Abaddon’s Gate (2013)	Cibola Burn (2014)
Brandon Sanderson (1975) -- fantasy	
The Way of Kings (2010)	Words of Radiance (2014)
Oathbringer (2017)	Rythm of War (2020)

JS - 1 - Implementace jednoduché 2D kamery a pohyblivých objektů

Pracovní úkoly:

1. Ve vaší webové stránce vytvořte „oblast“, kterou budeme dále označovat jako `viewport` – oblast reprezentující zorné pole kamery. Dále vytvořte oblast jako child `viewport`u, kterou označíme jako `scene` – oblast reprezentující prostor, ve kterém bude možné vykreslovat různé objekty jako HTML elementy.
2. Do oblasti `scene` umístěte minimálně dva různé objekty tak, aby jeden byl vidět, protože se nachází v oblasti určené `viewport`em a ten druhý nebyl vidět, protože se nachází mimo oblast určenou `viewport`em.
3. Vytvořte třídu `Camera`, která umožní uživateli pohybovat s oblastí `scene` pomocí myši takzvaným „dragováním“ – pokud uživatel přesune ukazatel myši na oblast `scene` a bude mít stisknuté dané tlačítko (například levé), pak se celá oblast `scene` začne posouvat společně s ukazatelem myši.
4. Vytvořte třídu `MovableObject`, pomocí které bude možné efektivně přidávat různé objekty do oblasti `scene`.
5. Modifikujte třídu `MovableObject` tak, aby uživateli umožnila měnit polohu vykresleného objektu v oblasti `scene` dragováním (viz pracovní úkol 3, ale pozor, implementace pro `MovableObject` se může lišit od implementace této funkce v třídě `Camera`).
6. Přidejte do vaší webové stránky tlačítko, po jehož stisknutí vznikne nový objekt třídy `MovableObject`, který se bude pohybovat společně s ukazatelem myši, dokud uživatel nestiskne dané tlačítko myši (opět například levé) v oblasti `scene`.

JS - 2 - Vykreslení modifikovatelného trojúhelníku

Pracovní úkoly:

1. Vytvořte `div` element určité šířky a výšky, ve kterém je možné vykreslit úsečku určenou souřadnicemi počátečního a koncového bodu.[5]
2. Element z pracovního úkolu 1 „zabalte“ do třídy `Node`, která bude reprezentovat stranu trojúhelníku.
3. Vytvořte třídu `Vertex`, která bude reprezentovat vrchol trojúhelníku pomocí vhodného HTML elementu.
4. Vytvořte třídu `Triangle`, která vykreslí trojúhelník pomocí tří objektů třídy `Vertex` a tří objektů třídy `Node`.
5. Rozšiřte třídu `Vertex` tak, aby uživatel mohl měnit polohu daného vrcholu pomocí myši takzvaným „dragováním“ -- pokud uživatel přesune ukazatel myši na daný vrchol a bude mít stisknuté dané tlačítko (například levé), pak se vrchol začne posouvat společně s ukazatelem myši. Ujistěte se, že po posunutí vrcholu uživatelem bude stále na webové stránce vykreslen trojúhelník – tedy že se po posunutí updatují i relevantní strany trojúhelníku.

[5] Doporučujeme do elementu `div` vložit element (pomocí „vlastnosti“ `innerHTML`) `<svg width="100%" height="100%" style="display:block; overflow: visible;"></svg>`, do kterého je následně možné vložit SVG element `<line />` (opět pomocí `innerHTML`), který již vykreslí požadovanou úsečku po zadání parametrů `x1`, `y1` pro počáteční bod a `x2`, `y2` pro koncový bod. Dalšími užitečnými parametry SVG elementu `line` jsou `stroke` – barva úsečky, a `stroke-width` – tloušťka úsečky.

JS - 3 - Simulace šikmého vrhu

Pomocný text: *Eulerova metoda*

Pracovní úkoly:

1. Rozdělte webovou stránku na dva sloupce, přičemž první (levý) sloupec si pro účely tohoto zadání označíme jako „řídící panel“ a druhý (pravý) jako „viewport“.
2. Řídící panel bude obsahovat takzvané „ovládací prvky“, pomocí kterých bude možné nastavit počáteční polohu (x, y) a rychlost (v_x, v_y) „míčku“ (viz pracovní úkol 3) a zároveň velikost tíhového zrychlení g. Nakonec bude obsahovat také prosté tlačítko „start“, které zastaví, nebo spustí simulaci (viz pracovní úkol 4). „Ovládací prvek“ vytvořte následovně:
   - **Musí se jednat o třídu**, která vytvoří potřebné HTML elementy: název ovládacího prvku, input a tlačítko. Zároveň by tato třída měla umožnit „programátorovi“ definovat vlastní feedback funkci pro tlačítko (konkrétní podobu této funkce definuje „programátor“ mimo samotnou třídu).

Název ovládacího prvku	
Input	Tlačítko

   - „Použití“ této třídy by **mohlo** vypadat například následovně:
     ```
     const xControl = new Control(…);
     xControl.button.onclick = (…) => {…};
     ```
3. Viewport bude obsahovat minimálně jeden HTML element – takzvaný „míček“, který je svázán s odpovídající třídou, která uchovává všechny relevantní fyzikální vlastnosti (minimálně):
   - Okamžitou polohu (x, y).
   - Okamžitou rychlost (v_x, v_y).
4. Vytvořte metodu `simulate(timestamp)`, ve které implementujete simulaci pomocí takzvané Eulerovy metody (viz pomocný text: *Eulerova metoda*) a na konci této metody vytvoříte animační cyklus (cyklus, který neblokuje ostatní procesy) pomocí příkazu: `animID = requestAnimationFrame(simulate);` Kde `animID` je globální proměnná, pomocí které budete moci animaci zastavit (viz pracovní úkol 5). Metoda `simulate()` automaticky přijímá parametr `timestamp`, který je roven `document.timeline.currentTime`, což reprezentuje uplynulý čas v ms.
5. Implementujte tlačítko start tak, aby se střídalo spouštění a zastavování/resetování simulace:
   - Spuštění simulace je možné zajistit pomocí následujícího příkazu:
     `requestAnimationFrame(simulate);`
   - Zastavení animace je možné zajistit pomocí příkazu:
     `cancelAnimationFrame(animID);`
     Kde `animID` je reference na poslední snímek uložená v globálním kontextu, jejíž hodnotu neustále přepisuje metoda `simulate()`.