TYPE SCRIPT SLABIKÁŘ

Obsah

Typ	Použití	Příklad použití
`number`	čísla	`let x: number = 42`
`string`	text	`let s: string = "ahoj"`
`boolean`	pravda/nepravda	`let b: boolean = false`
`any`	(nedoporučeno, cokoliv)	`let cokoliv: any`
`void`	návratová hodnota funkce	`function foo(): void {}`
`T[]`	pole hodnot	`let pole: string[]`
`type`	alias, jednoduché objekty	viz výše
`interface`	větší datové objekty, dědičnost	viz výše
`enum`	výčet několika možností	viz výše
`` `	`	„nebo“, unie více typů viz výše

Rychlý souhrn základních typů v TypeScriptu

number Celá a desetinná čísla

```
let skore: number = 0;
```

string Text

```
let jmeno: string = "Motyl";
```

boolean Pravda/nepravda

```
let jeZivy: boolean = true;
```

null a undefined Zvláštní hodnoty „nic“ a „nedefinováno“

```
let vybrano: string | null = null;
```

Pole (arrays)

`number[]`, `string[]`, `any[]`

```
let cisla: number[] = [1, 2, 3];
let slova: string[] = ["banán", "baterie"];
```

Objektové typy a interface

Objekt s vlastnostmi:

```
let odpadek: { typ: string; status: string; };
odpadek = { typ: "Banán", status: "default" };
```

type alias:

```
type Odpadek = { typ: string; status: string; };
let odpadky: Odpadek[] = [];
```

interface: (Vhodnější pro větší objekty, lze rozšiřovat a dědit)

```
interface Odpadek {
  typ: string;
  pozice: { x: number; y: number };
  status: string;
  sprite: Phaser.GameObjects.Sprite | null;
}
```

Použití const ve funkci/metodě (stejné jako v JS):

```
create(): void {
    const pytel = this.add.sprite(400, 300, 'pytel');
    // pytel je platný jen uvnitř create()
    // dál pracuješ s pytel (ne s this.pytel)
}
```

Výhoda: Proměnná je „jen pro tuto metodu“ – nemůžeš ji používat jinde v třídě. Typ si TypeScript odvodí automaticky (nemusíš často uvádět, ale můžeš):

```
const pytel: Phaser.GameObjects.Sprite = this.add.sprite(400, 300, 'pytel');
```

Rozdíl mezi *const pytel = ...* a *this.pytel = ...*:

const pytel: Použiješ jen ve funkci/metodě, kde ji vytvoříš (např. v create()). Nejčastější pro jednorázové objekty, které nepotřebuješ dál v celé třídě.

this.pytel: Potřebuješ-li k pytli přistupovat i v jiných metodách třídy (např. v update(), ve vlastních funkcích, při kolizích...), je lepší deklarovat jako vlastnost třídy.

Praktický příklad obou variant:

```
export default class GameScene extends Phaser.Scene {
    // Pro opakované použití v celé třídě:
    pytel!: Phaser.GameObjects.Sprite;

    create(): void {
        // Varianta 1: vlastnost třídy
        this.pytel = this.add.sprite(400, 300, 'pytel');

        // Varianta 2: pouze lokální proměnná
        const pytel = this.add.sprite(400, 300, 'pytel');
        // pytel platí jen uvnitř create()
    }

    update(): void {
        // přístup jen k this.pytel, ne k 'const pytel'
    }
}
```

Shrnutí: *const pytel = ...* – používej jen, když pytel nepotřebuješ jinde než v dané metodě. *this.pytel = ...* – když potřebuješ pytel používat napříč třídou/scénou.

Speciální typy pro Phaser

Phaser.GameObjects.Sprite Odkaz na sprite objekt (přes import z Phaseru) **void** Funkce nic nevrací

```
function vypis(): void { ... }
```

any Jakýkoli typ (používej co nejméně)

```
let cokoliv: any;
```

Další užitečné typy (enum, union)

enum Výčet možných hodnot

```
enum Stav { Default, Spravny, Spatny }
let aktualni: Stav = Stav.Default;
```

Union (|) Proměnná může být jeden z typů

```
let stav: "default" | "spravne" | "spatne";
```

Příklady pro hru v Phaseru

Alias typu pro pole odpadků:

```
type Odpadek = {
  typ: string;
  pozice: { x: number; y: number };
  status: string;
  sprite: Phaser.GameObjects.Sprite | null;
};

let odpadkyData: Odpadek[] = [];
```

Nejčastější typy v metodách Phaser scény

```
init(): void { ... }
preload(): void { ... }
create(): void { ... }
update(time: number, delta: number): void { ... }
```

Deklarace vlastností v těle třídy

Bez aliasu:

```
export default class Intro extends Phaser.Scene {
    odpadkyData: {
        typ: string;
        pozice: { x: number; y: number };
        status: string;
        sprite: Phaser.GameObjects.Sprite | null;
    }[];

    // ...
}
```

S type aliasem:

```
type Odpadek = {
    typ: string;
    pozice: { x: number; y: number };
    status: string;
    sprite: Phaser.GameObjects.Sprite | null;
};

export default class Intro extends Phaser.Scene {
    odpadkyData: Odpadek[];

    // ...
}
```

Sdílení typů napříč scénami

1. Vytvoř samostatný soubor s typem (např. `types/Odpadek.ts`):

```
export interface Odpadek {
    typ: string;
    pozice: { x: number; y: number };
    status: string;
    sprite: Phaser.GameObjects.Sprite | null;
}
```

2. Importuj typ v jiných souborech:

```
import { Odpadek } from '../types/Odpadek';

export default class Intro extends Phaser.Scene {
    odpadkyData: Odpadek[];

    // ...
}
```

Typické chyby a rady

- Pokud ti TypeScript píše, že vlastnost neexistuje v typu, musíš ji přidat do deklarace třídy.
- `string[]` a `Array<string>` znamená totéž, používej co ti vyhovuje.
- U vlastních typů používej raději interface, pokud budeš typ rozšiřovat na více místech.
- `any` používej jen výjimečně – přijdeš o výhody TypeScriptu!
- Čím víc budeš typovat, tím méně chyb později chytíš až při běhu hry.
- Pokud máš dlouhé typy, použij alias nebo interface, zjednodušíš si život.
- Kód si rozděluj na sekce a používej přehledné názvy (pro tebe i ostatní).

Web fonty Google

do **index.html** vložit:

```
<link href="https://fonts.googleapis.com/css2?
family=Roboto:wwght@700&display=swap" rel="stylesheet">
```

PreloadScene (TypeScript) – čekání na Google Font Loader

V PreloadScene používáme WebFont Loader, aby se fonty skutečně stáhly před zobrazením textů v další scéně.

```
// src/game/scenes/PreloadScene.ts

/// <reference path="../../types/webfontloader.d.ts" />

declare global {
  interface Window {
    WebFont: any;
  }
}

export default class PreloadScene extends Phaser.Scene {
  constructor() {
    super('PreloadScene');
```

```
  }

  preload(): void {
    // 1) Načtení WebFont Loaderu
    this.load.script(
      'webfont',
      'https://ajax.googleapis.com/ajax/libs/webfont/1.6.26/webfont.js'
    );

    // 2) Ostatní assety: obrázky, vlaječky, překlady…
    this.load.image('menu-bg',   'assets/images/menu-background.png');
    this.load.image('flag-cs',   'assets/images/flag-cs.png');
    this.load.image('flag-en',   'assets/images/flag-en.png');
    this.load.image('flag-pl',   'assets/images/flag-pl.png');

    this.load.json('locale-cs',  'assets/locales/cs.json');
    this.load.json('locale-en',  'assets/locales/en.json');
    this.load.json('locale-pl',  'assets/locales/pl.json');
  }

  create(): void {
    // 3) Počkáme na načtení fontu a až poté spustíme MainMenu
    window.WebFont.load({
      google: { families: ['Roboto:700'] },
      active: () => {
        // (volitelně) barva pozadí během čekání
        this.cameras.main.setBackgroundColor('#000000');
        // start next scene až fonty jsou načtené
        this.scene.start('MainMenu');
      }
    });
  }
}
```

Klíčové body this.load.script() – stáhne externí knihovnu WebFont Loader. declare global { interface Window { WebFont: any } } – dává TS vědět o existenci window.WebFont. window.WebFont.load({ ..., active: () => { ... } }) – callback active se vykoná, až jsou fonty k dispozici. this.scene.start('MainMenu') se volá až v active, takže v MainMenu už můžete bezpečně používat Google Fonts

Tvorba dialogových bublin ve Phaseru 3

Tento návod tě provede krok za krokem tvorbou lokalizovaných dialogových bublin (textových boxů) ve Phaseru 3 s TypeScriptem. Výsledný postup můžeš vložit do svého slabikáře.

---

1) Struktura projektu

Ve složce `src` vytvoř následující adresáře a soubory:

```
src/
├── assets/
│   └── locales/
│       ├── cs.json
│       ├── en.json
│       └── pl.json
├── scenes/
│   ├── PreloadScene.ts
│   └── IntroScene.ts
├── utils/
│   └── DialogManager.ts
└── main.ts
```

- **assets/locales**: JSON soubory s překlady pro češtinu, angličtinu a polštinu.
- **scenes/PreloadScene.ts**: Načte JSONy do cache.
- **scenes/IntroScene.ts**: Vytvoří a použije `DialogManager`.
- **utils/DialogManager.ts**: Třída pro vykreslování a ovládání bublin.

🔄 2) Načtení JSON překladů v PreloadScene

Ve `src/scenes/PreloadScene.ts` načteme všechny lokalizační soubory:

```
export default class PreloadScene extends Phaser.Scene {
  constructor() {
    super({ key: 'PreloadScene' });
  }

  preload(): void {
    // načteme JSONy z assets/locales
    this.load.json('lang_cs', 'assets/locales/cs.json');
    this.load.json('lang_en', 'assets/locales/en.json');
    this.load.json('lang_pl', 'assets/locales/pl.json');

    // ... další assety (obrázky, zvuky) ...
  }

  create(): void {
    // přechod do úvodní scény s výchozím jazykem
    this.scene.start('IntroScene', { locale: 'cs' });
  }
}
```

> **Poznámka:** JSON se uloží do cache pod klíčem `lang_<kód jazyka>` (např. `lang_cs`).

📦 3) Implementace `DialogManager.ts`

Vytvoř `src/utils/DialogManager.ts` s podporou vnořených klíčů (`"intro.title"`, `"dialog.ghostWelcome"`):

```
import Phaser from 'phaser';

type Locale = 'cs' | 'en' | 'pl';

interface Translations {
  [key: string]: any;
}

export default class DialogManager {
  private scene: Phaser.Scene;
  private locale: Locale;
  private texts: Record<Locale, Translations>;
  private container?: Phaser.GameObjects.Container;

  constructor(scene: Phaser.Scene, locale: Locale) {
    this.scene = scene;
    this.locale = locale;
    // načteme JSONy z cache
    this.texts = {
      cs: this.scene.cache.json.get('lang_cs') as Translations,
      en: this.scene.cache.json.get('lang_en') as Translations,
      pl: this.scene.cache.json.get('lang_pl') as Translations,
    };
  }

  // změní jazyk
  public setLanguage(locale: Locale): void {
    this.locale = locale;
  }

  // načte text podle vnořené cesty "klíč1.klíč2"
  private getNestedText(path: string): string {
    const parts = path.split('.');
    let curr: any = this.texts[this.locale];
    for (const p of parts) {
      if (curr?.[p] !== undefined) curr = curr[p];
      else return '[missing text]';
    }
    return typeof curr === 'string' ? curr : '[invalid key]';
  }

  // zobrazí bublinu s textem
  public show(key: string, x = 400, y = 300): void {
    // odstraníme předchozí bublinu (pokud je)
    if (this.container) this.container.destroy();

    const content = this.getNestedText(key);

    // grafika bubliny
```

```
    const bubble = this.scene.add.graphics();
    bubble.fillStyle(0xffffff, 1);
    bubble.lineStyle(2, 0x000000, 1);

    const padding = 10;
    const txt = this.scene.add.text(0, 0, content, {
      fontFamily: 'Arial',
      fontSize: '18px',
      color: '#000000',
      wordWrap: { width: 300 }
    });

    // spočteme rozměry
    const bounds = txt.getBounds();
    const width  = bounds.width  + padding * 2;
    const height = bounds.height + padding * 2;

    // vykreslíme obdélník a ocásek
    bubble.fillRoundedRect(0, 0, width, height, 8);
    bubble.strokeRoundedRect(0, 0, width, height, 8);
    bubble.fillTriangle(
      width/2 - 10, height,
      width/2 + 10, height,
      width/2,       height + 20
    );
    bubble.lineTriangle(
      width/2 - 10, height,
      width/2 + 10, height,
      width/2,       height + 20
    );

    // pozicujeme text a bublinu
    txt.setPosition(padding, padding);
    this.container = this.scene.add.container(
      x - width/2,
      y - height - 20,
      [ bubble, txt ]
    );
  }

  // skryje bublinu
  public hide(): void {
    this.container?.destroy();
    this.container = undefined;
  }
}
```

> **Poznámka:** Kód je kompletně okomentovaný, aby byl srozumitelný i pro začátečníka.

▶ 4) Použití v `IntroScene.ts`

```
import Phaser from 'phaser';
import DialogManager from '../utils/DialogManager';

export default class IntroScene extends Phaser.Scene {
  private dialog!: DialogManager;
  private locale!: 'cs' | 'en' | 'pl';

  constructor() {
    super({ key: 'IntroScene' });
  }

  init(data: { locale: 'cs' | 'en' | 'pl' }) {
    this.locale = data.locale;
  }

  create(): void {
    this.dialog = new DialogManager(this, this.locale);

    // zobrazíme nadpis
    this.dialog.show('intro.title', 400, 100);

    // po 2 s ukážeme další text
    this.time.delayedCall(2000, () => {
      this.dialog.show('intro.selectLang', 400, 200);
    });

    // pro ukázku: po dalších 2 s přepneme jazyk a zobrazíme uvítání ducha
    // this.dialog.setLanguage('en');
    // this.dialog.show('dialog.ghostWelcome', 400, 300);
  }
}
```

☑ Shrnutí kroků

1. **Struktura projektu**: vytvoř složky `assets/locales`, `scenes`, `utils`.
2. **PreloadScene**: načti `cs.json`, `en.json`, `pl.json` z `assets/locales`.
3. **DialogManager**: třída pro vykreslování lokalizovaných bublin s podporou vnořených klíčů.
4. **IntroScene**: vytvoř instanci `DialogManager` a voláním `show(...)` zobrazuj text.

Nyní můžeš do `assets/locales/cs.json` přidat své české texty. Až budeš chtít, připravím pro tebe i překlady do angličtiny a polštiny s ohledem na geocachingovou terminologii. 🎯