Vývojové metodiky

Techniky vývoje softwaru jsou postupy, které určují, jakým způsobem se vyvíjí software.

- ▶ Agilní metodika (Scrum)
- ▶ Vodopádová metodika
- ▶ Kanban

Rychlé prototypování

Proces pro vytvoření funkčního modelu projektu co nejrychleji, aby bylo možné testovat a iterovat nápady.

> **Note**
>
> V kontextu Unity to znamená vytvoření základní verze hry nebo aplikace, která zahrnuje pouze klíčové mechaniky a funkce.

- ▶ Rychlá iterace
- ▶ Postup

Pojmenování BEM

BEM = "Block Element Modifier"

Metodika pro pojmenování tříd v `HTML` a `CSS`.

> **Note**
>
> Pomáhá udržet váš kód organizovaný a snadno pochopitelný, a to i pro ostatní vývojáře, kteří se na váš kód podívají.

Příklad:

```
<div class="block"> <!-- Block -->
  <div class="block__element"> <!-- Element -->
  </div>
  <div class="block__element--modifier"> <!-- Element with modifier -->
```

```
  </div>
</div>
```

```
.block { ... }
.block__element { ... }
.block__element--modifier { ... }
```

- Block
- Element
- Modifier
- Syntax BEM
- Použití v kódu

Warning

Informace níže jsou pouze orientační!

- Webové aplikace
- Mobilní aplikace
- Počítačové aplikace
- Databázový vývoj
- Herní vývoj
- CI & CD
- Testování

- Modifikátory přístupu
- Složka 'runtimes' a multiplatformní nasazení
- Uvolnění zdrojů
- Volání funkcí z externích DLL

Interface = rozhraní

▶ ICloneable

Kolekce FIFO/LIFO

Určují pořadí, ve kterém jsou prvky přidávány a odebírány.

- Queue
- PriorityQueue
- Stack

Seznamy

Seznamy jsou kolekce prvků, které lze indexovat a efektivně upravovat.

Umožňují přidávání, odstraňování a přístup k prvkům na základě jejich indexu.

- List
- LinkedList

Slovníky

Slovníky jsou kolekce klíč-hodnota, které umožňují efektivní vyhledávání, přidávání a odstraňování prvků na základě klíče.

Každý klíč v slovníku je jedinečný a je spojen s jednou hodnotou.

- Dictionary
- SortedDictionary

Kolekce bez duplicit

Neumožňují ukládání duplicitních prvků

- HashSet
- Hashtable

Kolekce Tuple

Umožňuje ukládání prvků různých typů v jedné kolekci.

Každý prvek v Tuple je přístupný pomocí pevně daného pořadí.

- Tuple
- ValueTuple

Pozorovatelné kolekce

Upozorňují na změny prvků, což je užitečné pro sledování změn v reálném čase.

- ObservableCollection

Kolekce pouze pro čtení

Kolekce, které nelze měnit po jejich vytvoření, což zajišťuje jejich neměnnost a bezpečnost

- ReadOnlyCollection
- ReadOnlyDictionary

Neměnné kolekce

Nelze měnit po jejich vytvoření, což zajišťuje jejich neměnnost a bezpečnost

- ImmutableArray
- ImmutableList
- ImmutableDictionary
- Immutable HashSet
- Immutable SortedSet
- Immutable Queue
- ImmutableStack

Paměťové kolekce

Umožňují bezpečný přístup k paměti a manipulaci s ní

- Memory
- Span

Slabé reference

Umožňují udržovat odkazy na objekty bez zabránění jejich uvolnění garbage collectorem

- WeakReference

Kolekce pro více vláken

Jsou bezpečné pro použití ve více vláknech, což zajišťuje synchronizaci a bezpečnost dat

- ConcurrentQueue
- ConcurrentStack
- ConcurrentDictionary
- ConcurrentBag
- BlockingCollection

Atributy obsahují **Metadata**

Datové anotace

= System.ComponentModel.Annotations (namespace)

- Nejvíce používané anotace:
  - ▶ [Required]
  - ▶ [Range]
  - ▶ [MaxLength]
  - ▶ [MinLength]
  - ▶ [StringLength]
  - ▶ [RegularExpression]
  - ▶ [DataType]
  - ▶ [Display]

▶ Příklad
▶ Vlastní datová anotace

FileHelpers

> ⊗ **Important**
>
> Nepodporuje:
>
> Záznamy s proměnnou délkou (každý záznam musí mít stejný počet polí)
>
> Změnu formátu za běhu (každý záznam musí mít stejný formát po celou dobu běhu programu)

- Nejvíce používané atributy:

Třída

- ▶ [DelimitedRecord]
- ▶ [FixedLengthRecord]

Pole

- ▶ [FieldTrim]

- ▶ [FieldOptional]
- ▶ [FieldIgnore]
- ▶ [FieldConverter]
- ▶ [FieldOrder]
- ▶ [FieldQuoted]

▶ Příklad

Vlastní konvertor

1. Vytvořit třídu a rozšířit ji o třídu `ConverterBase`.
2. Přepsat metody `StringToField` a `FieldToString.`

- Příklad:

  - ▶ Definice
  - ▶ Použití

Enum

= `enum` je výčtový typ, který umožňuje definovat vlastní datový typ, který může nabývat jedné z předem definovaných hodnot.

▶ Typy

- Základní pojmy
- Druhy metod
- Ukazetel na metody
- Asynchronní a Paralelní metody
- Task Parallel Library (TPL)
- Tipy

Implicitní a Explicitní operátory

Rozdíly:

- Implicitní:

  Automatický převod (žádný `cast` není potřeba)

- Explicitní:

  Vyžaduje `cast` (musíte převod jasně napsat)

> **Note**
>
> Implicitní je pohodlnější, ale explicitní je bezpečnější pro složité nebo nejednoznačné převody.

▶ Implicitní operátor
▶ Explicitní operátor

Vytvoření API

Vlastní REST API v C# s využitím frameworku `ASP.NET Core`.

- ▶ Vytvoření projektu
- ▶ Struktura projektu
- ▶ Přidání kontroleru
- ▶ Konfigurace závislostí

- Náhrada znaků
- CDATA
- Serializace a Deserializace objektu
- Namespace
- Konvence serializace XML

NUnit

= Testovací framework

▶ Multiple Asserts

Balíčky

- Globální balíčky
- Záloha
- Obnova

Python

Balíčky

- Záloha balíčků
- Instalace balíčků ze zálohy

WPF (Windows Presentation Foundation)

- Tvorba desktopových aplikací na platformě Windows
- Odděluje logiku aplikace (C#) od vzhledu (XAML)
- Umožňuje datové vazby a stylování
- Podpora vektorové grafiky, animací a multimédií

Prvky

▶ Button
▶ TextBox
▶ CheckBox
▶ ComboBox
▶ RadioButton
▶ Slider
▶ Vlastní ovládací prvek

Styly

Styl se používá k definování vzhledu a chování více prvků najednou.

Definuje se pomocí **XAML**.

▶ Definování stylu
▶ Použití stylu

Prefixy

x:

- Vyhrazen pro XAML standardní funkce a typy.
- Používá se pro přístup k základním vlastnostem, jako jsou `x:Class`, `x:Name`, `x:Key`, atd.
- Příklad

```
<Window x:Class="MyNamespace.MainWindow"
      x:Name="mainWindow"
      x:Key="myWindowKey">
```

xmlns:

- Používá se k deklaraci namespace.

- Obvykle se používá v kořenovém prvku XAML souboru.
- Příklad deklarace namespace:

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:MyNamespace">
```

local:

- Používán k odkazování na vlastní namespace aplikace.
- Můžete ho použít k přístupu k vlastním ovládacím prvkům, datovým modelům a dalším třídám definovaným ve vaší aplikaci.
- Příklad:

```
<local:MyCustomButton Content="Moje vlastní tlačítko"/>
```

xmlns:sys:

- Pro přístup k základním typům .NET, jako jsou `System.String`, `System.Int32`, atd.
- Příklad:

```
xmlns:sys="clr-namespace:System;assembly=mscorlib"
```

xmlns:controls:

- Pro přístup k ovládacím prvkům z externích knihoven, jako je například **Windows Community Toolkit**.
- Příklad:

```
<controls:MyCustomControl/>
```

xmlns:mc:

- Používá se pro **Markup Compatibility**.
- Umožňuje použití starších XAML formátů a zajišťuje zpětnou kompatibilitu.
- Příklad:

```
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
```

xmlns:d:

- Používá se pro návrhové časové funkce a umožňuje definovat prvky, které se zobrazují pouze během návrhu.
- Příklad:

```
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

Použití prefixů v XAML

```
<Window x:Class="MyNamespace.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:MyNamespace"
        xmlns:controls="clr-
namespace:MyCustomControls;assembly=MyCustomControlsAssembly"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        mc:Ignorable="d"
        Title="Hlavní okno" Height="350" Width="525">

    <Grid>
        <local:MyCustomButton Content="Moje vlastní tlačítko"
Width="200" Height="50"/>
        <Button Content="Tlačítko" Width="100" Height="30"/>
    </Grid>
</Window>
```

Šablony (ControlTemplates)

Šablony umožňují plně přizpůsobit vzhled ovládacího prvku.

Šablona definuje strukturu a vzhled prvku.

- Vytvoření šablony pro tlačítko

```
<Window.Resources>
  <ControlTemplate x:Key="MyButtonTemplate" TargetType="Button">
      <Border Background="{TemplateBinding Background}"
              BorderBrush="Black"
              BorderThickness="2"
              CornerRadius="5">
          <ContentPresenter HorizontalAlignment="Center"
                            VerticalAlignment="Center"/>
```

```
        </Border>
    </ControlTemplate>
</Window.Resources>
```

- **ControlTemplate**: Určuje, jak bude tlačítko vypadat.
- **TemplateBinding**: Slouží k vázání vlastností stylu na vlastnosti šablony.

- Použití šablony

```
<Button Template="{StaticResource MyButtonTemplate}"
        Background="LightBlue"
        Content="Stylizované tlačítko"/>
```

Responzivní design prvků

Responzivní design znamená, že se aplikace přizpůsobí různým velikostem a rozlišením obrazovky.

- ▶ Layout Panely
- ▶ Dynamické Velikosti
- ▶ Sledování Změny Velikosti
- ▶ ViewBox

Triggery

Triggery umožňují dynamicky měnit vzhled prvku na základě určitých událostí nebo podmínek.

- Použití triggeru

  Zde je příklad stylu tlačítka, který mění barvu pozadí, když je kurzor myši nad tlačítkem:

```
<Style TargetType="Button">
    <Setter Property="Background" Value="Gray"/>
    <Setter Property="Foreground" Value="White"/>
    <Style.Triggers>
        <Trigger Property="IsMouseOver" Value="True">
            <Setter Property="Background" Value="Green"/>
        </Trigger>
    </Style.Triggers>
</Style>
```

Data Binding (Vazba Modelu na View)

- 1. Vytvoření ViewModel
- 2. Vytvoření XAML pro UI
- 3. Nastavení DataContext

Validace

- INotifyPropertyChanged + IDataErrorInfo
- INotifyPropertyChanged + INotifyDataErrorInfo

Animace

WPF podporuje animace, které umožňují měnit vlastnosti prvků v čase.

Zde je příklad, jak animovat změnu barvy pozadí tlačítka, když na něj najedete:

```
<Button Content="Klikni na mě">
    <Button.Triggers>
        <EventTrigger RoutedEvent="Button.MouseEnter">
            <BeginStoryboard>
                <Storyboard>
                    <ColorAnimation Storyboard.TargetProperty="
(Button.Background).(SolidColorBrush.Color)"
                                    To="Red" Duration="0:0:1"/>
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger>
    </Button.Triggers>
</Button>
```

Flutter

Používá se pro vývoj mobilních aplikací pro Android a iOS.

Využívá **Dart** jako programovací jazyk.

Instalace a vytvoření nového projektu

- ▶ Instalace
- ▶ Vytvoření nového projektu

Záloha a obnova

- ▶ Záloha závislostí
- ▶ Obnova závislostí

Základní znalosti

- ▶ Widget

Příkazy

- ▶ Tabulka s příkazy

Balíčky

- ▶ Lokalizace (interní knihovna)

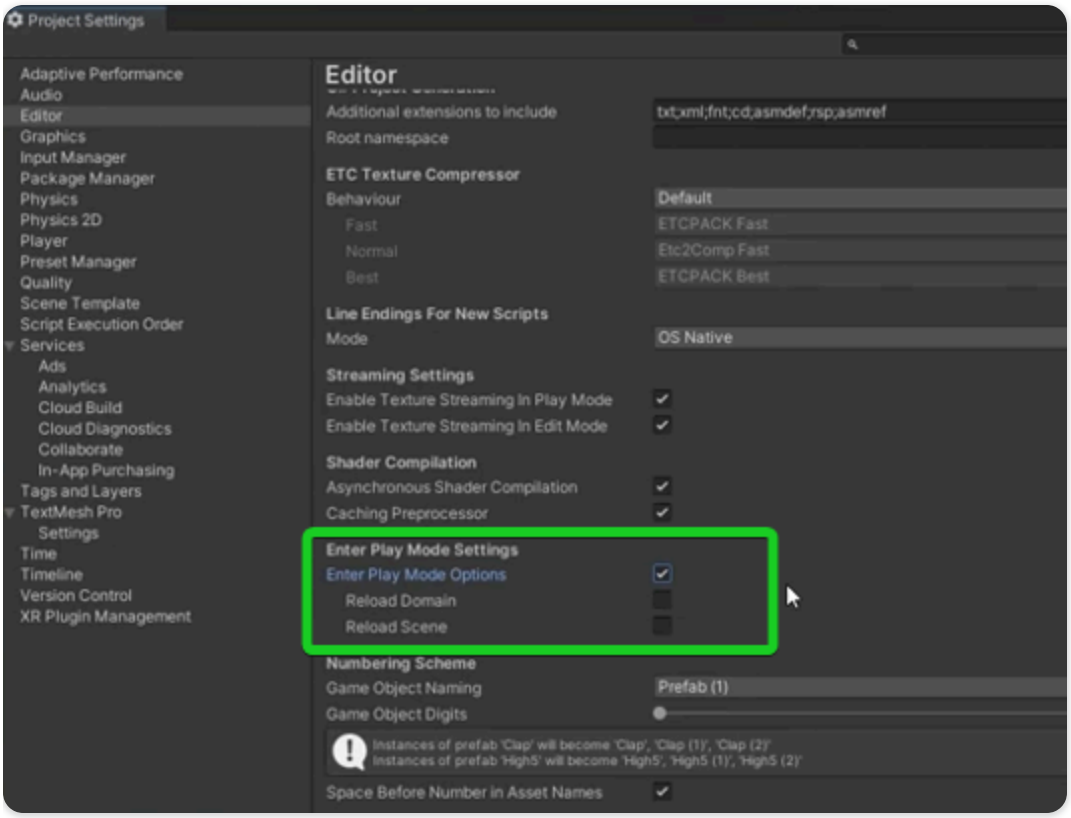
Pokrytí kódů (Code Coverage)

- ▶ Bez nahlédnutí do zdrojového kódu
- ▶ S nahlédnutím do zdrojového kódu

Řešení problémů

- ▶ Building with plugins requires symlink support.

Rychlejší spuštění



- Reload Domain

  Když je tato možnost povolena, všechny skripty se znovu načtou, což může trvat déle, ale zajišťuje, že se všechny změny v kódu projeví.

- Reload Scene

  Když je tato možnost povolena, Unity znovu načte aktuální scénu, což může být užitečné, pokud chceš začít s "čistým" stavem.

Pokud tyto možnosti zakážeš, můžeš zrychlit vstup do režimu hry, protože Unity se vyhne některým časově náročným procesům.

Výběr hry 2D či 3D

2D or 3D Unity Dev - What's better??

Rychlé prototypování

Prototyping Games in Unity?

Klíčová slova

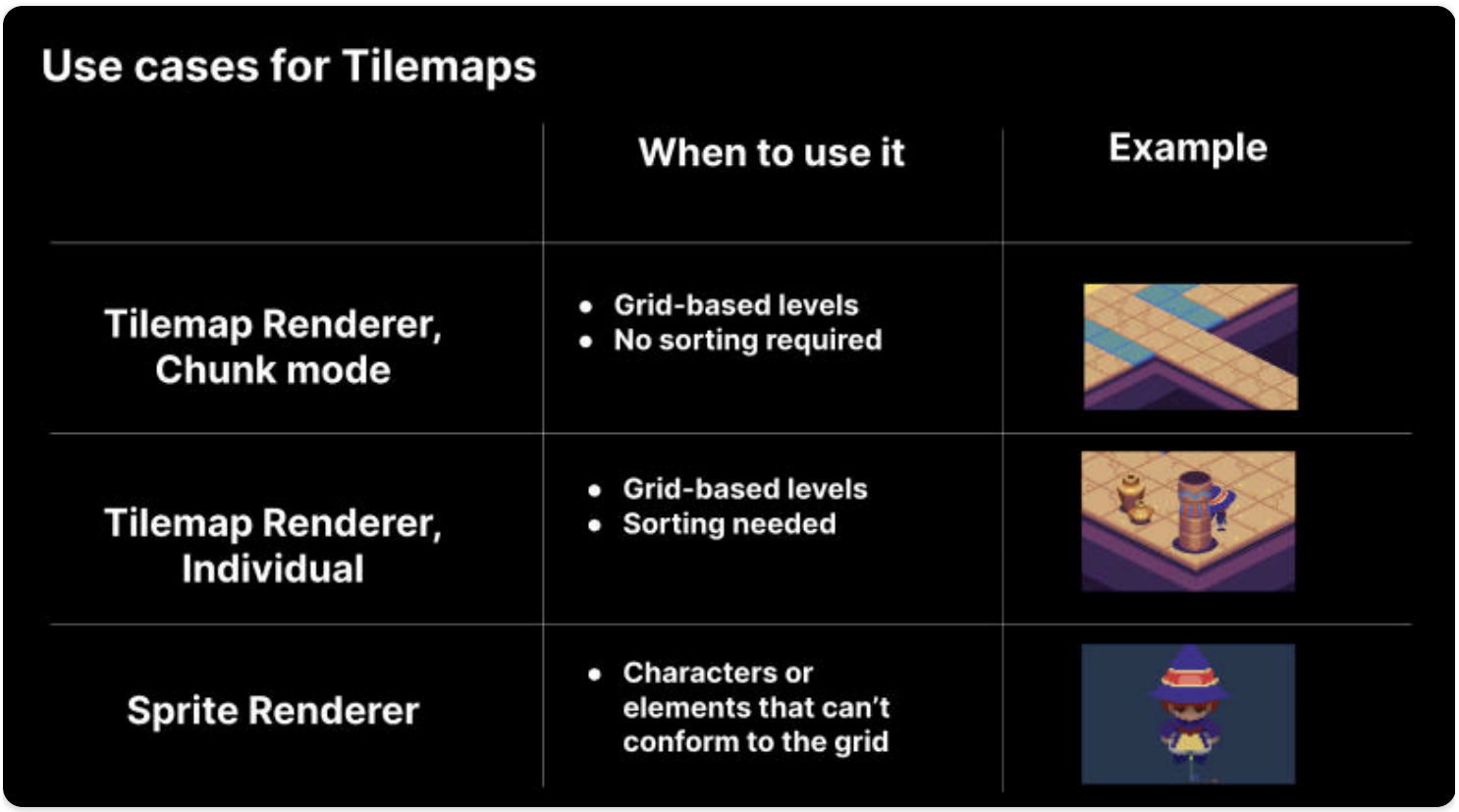
2 Keywords I didn't fully understand when starting gamedev...

Vývojové vzory

3 Game Programming Patterns WE ACTUALLY NEED.

2D

Tilemap x Sprite Renderer



Tilemap

- ▶ Vykreslit a nastavit barvu na dlaždici
- ▶ Pravidla pro Tilemap

Velikost obrázku

- ▶ Definice velikostí
- ▶ Nastavení velikosti

Animace obrázku

- ▶ Hloubka (Depth) u kostí
- ▶ Univerzální Rigging

Řešení chyb při vykreslování sprite

Important

Na kameře musí být přidána komponenta `Pixel Perfect Camera` pro 2D

Zabraňuje deformacím, rozmazání a trhání obrazu

- ▶ Černé čáry
- ▶ Problikávání

UMotion

▶ Uložení změn

Kamera

> **Important**
>
> Pro 2D hry musí být přidána komponenta "Pixel Perfect Camera", která zabrání deformacím, trhání obrazu atd..

- Ortografická Kamera
- Perspektivní Kamera
- Novinky

Navigační systém pro pohyb

▶ Novinky

Skriptovatelné Objekty

Nemusí se vytvářet ve scéně, jsou namísto toho vytvořeny již v projektu.

> **Important**
>
> ScriptableObjects se po zavření a opětovném otevření hry obnoví na výchozí hodnoty.

▶ Singleton

UI

Původní systém pro vytváření uživatelského rozhraní v Unity.

▶ Tlačítko (Button)

UI Toolkit

UI Toolkit je nový systém pro vytváření uživatelského rozhraní v Unity.

▶ Novinky

URP

= Universal Renderer Pipeline

- ▶ DefaultVolumeProfile
- ▶ UniversalRenderPipelineGlobalSettings
- ▶ URP Render Pipeline Asset
- ▶ URP Renderer Data

.NET CLI (Command Line Interface)

> **Important**
>
> Je zapotřebí mít nainstalovaný `.NET SDK (Software Development Kit)` a `.NET Runtime (Framework)`

- Umístění balíčků
- Seznam nainstalovaných balíčků
- Záloha globálních nástrojů
- Obnova globálních nástrojů

Příkazy

- Instalace
- Aktualizace
- Odinstalace

Nuget Packages

- Správa balíčků
- Globální složka balíčků

Appcast Feed XML

Appcast je RSS feed ve formátu XML.

Poskytuje informace o dostupných verzích aplikace.

Podporuje aktualizace aplikací pomocí technologie `Sparkle`.

Struktura:

- ▶ Hlavní komponenty
- ▶ Příklad feedu

Hlavni komponenty:

- ▶ <rss>
- ▶ <channel>
- ▶ <item>

Delta aktualizace:

> **Warning**
>
> Delta soubory je zapotřebí vytvořit pomocí nástroje pro generování delta souborů.

Delta soubory obsahují pouze rozdíly mezi verzemi aplikace.

Tento typ souboru šetří šířku pásma a urychluje proces aktualizace.

- ▶ Příklad