Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Využití návrhových vzoru ve vývoji herních aplikací

Martin Khol

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Martin Khol

Informatika

Název práce

Využití návrhových vzorů ve vývoji herních aplikací

Název anglicky

Use of design patterns in the game applications development

Cíle práce

Bakalářská práce je zaměřena na hledání vhodných návrhových vzorů pro vývoj herních aplikací. Hlavním cílem práce je vhodné návrhové vzory identifikovat a vytvořit odpovídající doporučení pro jejich případy využití. Dílčím cílem práce je vytvoření prototypu herní aplikace, ve kterém jsou vybrané návrhové vzory demonstrovány.

Metodika

Metodika zpracování teoretické části práce bude vycházet ze studia odborných informačních zdrojů. Na základě syntézy zjištěných informací budou vybrány konkrétní návrhové vzory jako východiska pro realizaci praktické části bakalářské práce.

Praktická část bude zaměřena na vývoj aplikace v herním engine Unity za použití programovacího jazyku C#. Nejdříve budou stanoveny vlastnosti a funkce požadovaného prototypu, které budou následně implementovány.

Na základě poznatků získaných z literární rešerše a využití prototypu budou návrhové vzory zhodnoceny a porovnány s alternativními přístupy.

Doporučený rozsah práce

35-40 stran

Klíčová slova

návrhové vzory, vývoj SW, Unity, C#, hry

Doporučené zdroje informací

AMPATZOGLOU, Apostolos a Alexander CHATZIGEORGIOU, 2007. Evaluation of object-oriented design patterns in game development. Information and Software Technology. 49(5), 445-454. ISSN 09505849. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S0950584906000929 doi:10.1016/j.infsof.2006.07.003

GAMMA, Erich, Richard HELM, Ralph JOHNSON a John VLISSIDES, c1995. Design patterns: elements of reusable object-oriented software. Boston: Addison-Wesley. Addison-Wesley professional computing series. ISBN 978-0201633610

NYSTROM, Robert, 2014. Game Programming Patterns. Seattle: Genever Benning. ISBN 978-0990582908.

QASIM, Awais, Adeel MUNAWAR, Jawad HASSAN a Adnan KHALID, 2021. Evaluating the Impact of Design Pattern Usage on Energy Consumption of Applications for Mobile Platform. Applied Computer Systems. 26(1), 1-11. ISSN 2255-8691. Dostupné z: https://www.sciendo.com/article/10.2478/acss-2021-0001 doi:10.2478/acss-2021-0001

WEDYAN, Fadi a Somia ABUFAKHER, 2020. Impact of design patterns on software quality: a systematic literature review. IET Software. 14(1), 1-17. ISSN 1751-8806. Dostupné z: https://onlinelibrary.wiley.com/doi/10.1049/iet-sen.2018.5446 doi:10.1049/iet-sen.2018.5446

Předběžný termín obhajoby

2021/22 LS – PEF

Vedoucí práce

Ing. Tomáš Benda

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 1. 11. 2021

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 23. 11. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 13. 03. 2022

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Návrhové vzory a jejich využití v herních aplikacích" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.3.2021 _____

Poděkování

Rád bych touto cestou poděkoval Panu Ing. Tomáši Bendovi za odborné vedení této práce.

Využití návrhových vzorů ve vývoji herních aplikací

Abstrakt

Tématem této bakalářské práce je využití návrhových vzorů v oblasti vývoje herních aplikací a vytvoření seznamu vhodných zdrojů. Dílčím cílem je vývoj herní aplikace, která návrhové vzory využívá.

Teoretická část práce se zaobírá obecným popisem vybraných návrhových vzorů. V každé kapitole je rozebrána jejich implementace pomocí UML diagramu tříd. Představeny jsou také výhody, nevýhody a případy využití návrhových vzorů. Závěrem teoretické části je představení herního engine Unity a základních pojmů pro práci v něm.

V praktické časti je předveden návrh herní aplikace, kde jsou vysvětleny jednotlivé herní prvky. Vývoj aplikace je následně realizován pomocí návrhových vzorů popsaných v teoretické části. Každé použití návrhového vzoru je detailně vysvětleno.

V závěru práce jsou stanoveny výhody a nevýhody využití návrhových vzorů ve vypracované aplikaci a obecně pro oblast herního vývoje.

Klíčová slova: návrhové vzory, vývoj SW, Unity, C#, hry

Use of design patterns in the development of game applications

Abstract

The topic of this bachelor thesis is the use of design patterns in the development of game applications and the creation of a list of suitable resources. A partial goal is the development of a game application that uses design patterns.

The theoretical part of the thesis contains a general description of the selected design patterns. Each chapter discusses their implementation using a UML class diagram. The advantages, disadvantages and use cases of design patterns are also presented. Last chapter of the theoretical part is the introduction of the game engine Unity and basic concepts for working in it.

The practical part presents the design of a game application, where the individual game elements are explained. The development of the application is influenced by the design patterns in the theoretical part. The use of the design pattern is explained in detail.

At the end of the thesis the advantages and disadvantages of using design patterns in the developed application and in general for the field of game development are discussed.

Keywords: design patterns, software development, Unity, C#, games

Obsah

Seznam obrázků

1 Úvod

Vývoj software je široká oblast a jednou z jeho součástí je potřeba organizace struktury kódu. Dobrá organizace vede k větší přehlednosti, robustnosti a jednodušší znovupoužitelnosti kódu. Tohoto lze dosáhnout dodržováním ověřených zásad a praktik jakými jsou i návrhové vzory.

Návrhové vzory obecně jsou doporučené postupy pro řešení často se vyskytujících úloh. Původně pochází tento pojem z architektury (Alexander, 1977). Ve vývoji software se začal objevovat spolu s popularizací objektově orientovaného přístupu k návrhu software. Jejich hlavním přínosem je zefektivnění návrhu počítačových programů a zjednodušení jejich implementace.

V kontextu herního vývoje existují specifické návrhové vzory, které nejsou běžné v jiných oblastech návrhu software, ale jsou přítomné ve většině herních aplikací na dnešním trhu. Dnešní herní enginy využívají mnoho návrhových vzorů a při jejich používání je důležité jím porozumět pro jejich optimální využití.

Tato práce je věnována právě tématu návrhových vzorů. Cílem je vytvoření příručky nejdůležitějších návrhových vzorů pro začínající vývojáře. V teoretické části uvede tato práce několik vybraných návrhových vzorů vhodných pro návrh herních aplikací a uvede jejich výhody a využití. Tyto vzory byly vybrány na základě studia uvedených zdrojů.

V praktické části práce následuje návrh a implementace herní aplikace v herním engine Unity, která prakticky demonstruje využití vybraných návrhových vzorů.

2 Cíl práce a metodika

2.1 Cíl práce

Bakalářská práce je zaměřena na hledání vhodných návrhových vzorů pro vývoj herních aplikací. Hlavním cílem práce je vhodné návrhové vzory identifikovat a vytvořit odpovídající doporučení pro jejich případy využití. Dílčím cílem práce je vytvoření prototypu herní aplikace, ve kterém jsou vybrané návrhové vzory demonstrovány.

2.2 Metodika

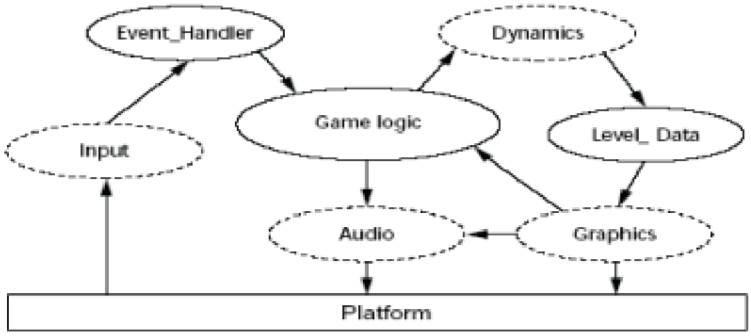
Metodika zpracování teoretické části práce bude vycházet ze studia odborných informačních zdrojů. Na základě syntézy zjištěných informací budou vybrány konkrétní návrhové vzory jako východiska pro realizaci praktické části bakalářské práce. Praktická část bude zaměřena na vývoj aplikace v herním engine Unity za použití programovacího jazyku C#. Nejdříve budou stanoveny vlastnosti a funkce požadovaného prototypu, které budou následně implementovány. Na základě poznatků získaných z literární rešerše a využití prototypu budou návrhové vzory zhodnoceny a porovnány s alternativními přístupy.

3 Herní vývoj

Herní aplikace se během posledních let rozrostly v jedno z nejvýdělečnějších odvětví softwarového průmyslu (Ampatzoglou, 2007). Hry se staly velice důležitou součástí našich životů.

Vývoj her je velice široká a komplexní oblast, kde návrhové vzory hrají důležitou roli v kontextu jejich návrhu a vývoje. Návrh a vývoj velkých aplikací vyžaduje mnoho hodin lidské práce. Pro zjednodušení tohoto problému, bývají aplikace rozděleny na jednotlivé systémy, na které se dá nahlížet jako na samostatné nezávislé moduly.

Návrhové vzory hrají důležitou roli v umožnění této abstrakce. Využíván je například vzor Composite [kapitola 4.5]. Ten umožňuje hierarchické skládání objektů z komponent, kde každá komponenta může představovat jeden modul. Například komponenta vykreslování grafické reprezentace objektu, přehrávání zvukových efektů nebo zpracování uživatelského vstupu.



Obrázek 1 Obecná architektura herní aplikace (Bishop, 1998)

4 Návrhové vzory

Pojem návrhové vzory pochází původně z architektury. Ta ho používá pro ‚společné řešení běžných problémů' (Alexander, 1977). Podobný význam mají návrhové vzory v architektuře softwaru. Pokud jsou využity správně, zvyšují flexibilitu a znuvupoužitelnost kódu vnitřního systému. Objektově orientované vzory specifikují vztahy mezi zúčastněnými třídami a určují jejich kolaboraci. Účelem toho řešení je zlepšení přizpůsobivosti modifikací původního návrhu za účelem ulehčení budoucích změn (Gamma, 1995).

V dnešní době je většina herních aplikací vyvíjena pro mobilní zařízení (Kevuru Games, 2022). Proto je důležité uvažovat při návrhu software také o jeho spotřebě elektrické energie. Vhodné využití některých návrhových vzorů může vést k nižší spotřebě (Qasim, 2021).

Je složité najít literaturu, která by byla komplexním seznamem všech používaných návrhových vzorů. Takový seznam by se dal využívat pro snadnější komunikaci mezi vývojáři a zpřehlednil dokumentaci. Z tohoto důvodu se tato práce věnuje vytvoření seznamu vzorů pro práci v herním vývoji. Na základně studie literatury, webových a vědeckých článků byly vybrány návrhové vzory vhodné pro využití při vývoji herních aplikací. Vybrány byly tyto vzory:

- Singleton
- Observer
- Strategy
- Factory
- Composite
- Game loop
- Update
- Object pool

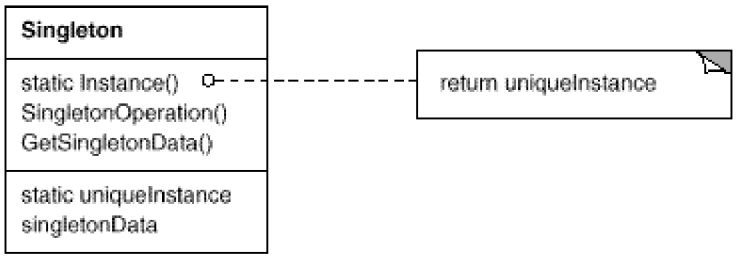
Následující kapitoly krátce uvedou jednotlivé vzory. Jejich implementace je znázorněna pomocí UML diagramů tříd. Závěrem každé kapitoly je diskuse výhod a nevýhod využití jednotlivých vzorů.

4.1 Singleton

4.1.1 Účel vzoru

Základním paradigmatem objektově orientovaného programování je, že každá třída umožňuje vytvořit libovolné množství na sobě nezávislých instancí. V určitých situacích, ale je zapotřebí mít tuto instanci jen jednu a zároveň musí být snadno dostupná. Globální proměnná umožní snadnou dostupnost, ale nezajistí, aby objekt neměl více instancí. Lepším řešením tohoto problému je předat tuto zodpovědnost přímo konkrétní třídě. Tato třída zajistí, aby nebylo možno vytvořit nové instance (přerušením žádostí o jejich vytvoření) a zajistí přístup k její jediné instanci. Toto umožňuje vzor Singleton. (Gamma, 1995); (Pecinovský, 2007)

4.1.2 Diagram



Obrázek 2 Diagram tříd vzoru Singleton (Gamma, 1995)

Při vytvoření objektu využívajícího návrhový vzor Singleton je zajištěna jeho unikátnost kontrolou, zda již takovýto objekt existuje. Objekt je globálně dostupný v rámci celého jmenného prostoru pomocí statické funkce Instance, která vrací odkaz na jeho jedinou instanci. Dále je možné s objektem libovolně pracovat. Objekt může obsahovat data nebo funkce, které musí být globálně přístupné. (Pecinovský, 2007)

4.1.2.1 Zajištění jediné instance

Návrhový vzor Singleton vytváří jednu normální instanci třídy, ale tato třída je napsaná tak, že pouze jedna jediná instance může být vytvořena (Gamma, 1995). Běžný způsob, jak tohoto lze docílit, je skrytím konstruktoru uvnitř metody třídy. Ta garantuje vytvoření jediné instance. Tato metoda má přístup k proměnné, která obsahuje unikátní instanci a zajišťuje její vytvoření před tím, než ji vrátí jako návratovou hodnotu. V jazyce C# je tato funkce definována jako statická a proměnná s odkazem na unikátní instanci také. (Gamma, 1995)

```
public class Singleton
{
    public static Singleton Instance()
    {
        if (_instance == null)
        {
            _instance = new Singleton();
        }
        return _instance;
    }

    protected Singleton() { }

    private static Singleton _instance;
}
```

Obrázek 3 Implementace vzoru Singleton v jazyce C#

4.1.3 Výhody a nevýhody

Singleton patří mezi vzory, které jsou snadné na pochopení. Kdykoli je potřeba přístup k datům nebo funkcionalitě, které nemají v dané aplikaci více instancí, je vhodné použít Singleton. Například pokud je v aplikaci právě jeden hráč jedno uživatelské rozhraní a jedna kamera. Využití vzoru Singleton zabrání vytvoření dalších nechtěných instancí a zajistí k objektům globální přístup.

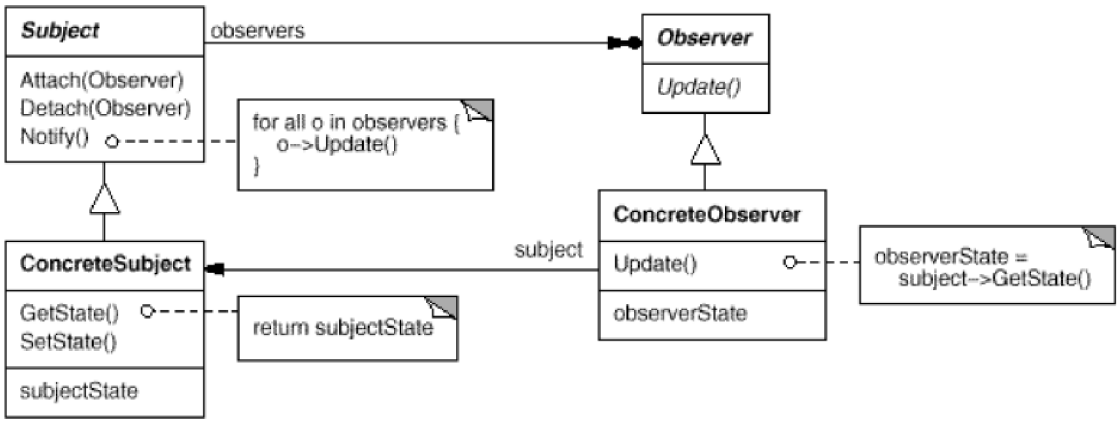
V závislosti na velikosti projektu je ale zapotřebí být s využitím vzoru Singleton, podobně jako s globálními proměnnými, opatrný. V případě, že bude nutné najít chybu a porozumět změnám dat, Singletony mohou, kvůli neomezenému přístupu ostatních tříd, porozumění chybám velice ztížit a kód udělat méně přehledným. (Grudl, 2008)

4.2 Observer

4.2.1 Účel vzoru

Vedlejším efektem rozdělení systému na spolupracující objekty je potřeba udržet konzistenci mezi závislými objekty. Nemělo by se tohoto dosahovat pomocí pevných vazeb, protože to by omezovalo jejich možnost opětovného použití. Změna stavu jednoho objektu může vyvolat reakce v mnoha dalších objektech. K tomuto druhu jednostranné závislosti lze využít vzor Observer. Tento návrhový vzor se využívá například pro nezávislost grafického interface a aplikačních dat. To umožňuje změnit implementaci aplikační logiky, aniž by bylo potřeba upravovat grafický interface. (W3sDesign, 2022)

4.2.2 Diagram



Obrázek 4 UML Diagram tříd vzoru Observer (Gamma, 1995)

- Třída Subject – Představuje interface objektu, kterého změna vyvolá reakci. Obsahuje seznam objektů, které následně notifikuje při změně stavu.
- Třída Observer – Definuje interface objektu, který je informovaný o změnách stavu objektu Subject. Může se k objektu Subjekt připojit a také se odpojit.
- Třída ConcreteSubject – Představuje konkrétní objekt, kterého stav je pozorován.
- Třída ConcreteObserver – Implementuje interface Observer pro komunikaci s objektem Subject a implementuje funkci Update, která udržuje jeho stav konzistentní se stavem Subjectu. (Gamma, 1995)

4.2.3 Výhody a nevýhody

Z vlastností tohoto návrhového vzoru je vhodné zdůraznit:

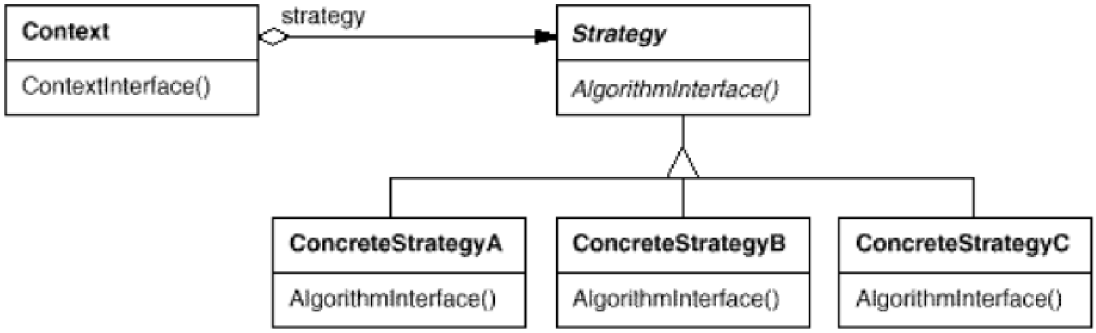
1. Abstraktní vazby – Objekty typu Subjekt znají pouze seznam objektů, které dědí z typu Observer. Nezáleží jím na jejich konkrétní implementaci. Jelikož tyto dva objekty nejsou jinak vázány a fungují nezávisle, umožňují rozdělení systému na nezávislé domény. Pokud dojde ke změně ve třídě využívající interface Observer nebo k jeho úplnému nahrazení, není potřeba provádět žádné úpravy ve třídě Subject.
2. Možnost hromadného volání – Nezáleží na množství objektů s interface Observer, které jsou přihlášené k notifikacím o změnách objektu Subject. To umožňuje libovolně přidávat a odebírat objekty Observer.
3. Neočekávané volání Update – Vzhledem k tomu, že objekty, které mění stav objektu Subject, nemají představu o tom, jaké objekty jsou vázáné na notifikaci o změně stavu, mohou zdánlivě nevýznamné změny vyvolat kaskádu změn Observerů a dalších na nich závislých objektů. Špatně definované závislosti objektů mohou vést k chybám, které je těžké nalézt. (Gamma, 1995)

4.3 Strategy

4.3.1 Účel vzoru

Vzor Strategy umožňuje definovat skupinu algoritmů, které jsou vzájemně zaměnitelné. Jejich záměna je možná za běhu programu a je nezávislá na klientech, kteří tento algoritmus používají. Algoritmus, který se chová odlišně v jiných podmínkách, lze pro zjednodušení rozdělit na několik algoritmů. V konkrétních situacích se využívá jen ten potřebný. (Freeman, 2014)

4.3.2 Diagram



Obrázek 5 Diagram tříd vzoru Strategy (Gamma, 1995)

- Třída Strategy – Definuje obecný interface pro všechny podporované algoritmy.
- Třída Context – Má nastaven odkaz na konkrétní objekt Strategy. Může také obsahovat interface pro přístup algoritmu k jeho datům.
- Třída ConcreteStrategy – Implementuje konkrétní algoritmus podle interface Strategy. (Gamma, 1995)

4.3.3 Výhody a nevýhody

Díky využití objektové kompozice nedochází při změně algoritmu ke změně vnějšího objektu, a ten zůstává nezměněn pro všechny ostatní objekty. Pro tyto situace, ve kterých je potřeba jednotný přistup k objektům a které se mohou chovat uvnitř rozdílně, je tento vzor vhodný.

Jeho využití umožňuje vyhnout se komplikovaným podmínkám. Návrhový vzor deleguje zodpovědnost za správné chování přímo na objekt Strategy. Například bez vzoru Strategy by objekt určující chování charakteru mohl vypadat takto:

```
void Update()
{
    switch (behaviour)
    {
        case Behaviour.Passive:
            DoPassiveBehaviour();
            break;
        case Behaviour.Aggressive:
            DoAggresiveBehaviour();
            break;
        default:
            break;
    }
}
```

Obrázek 6 Přístup bez využití vzoru Strategy

Návrhový vzor Strategy eliminuje tuto podmínku switch delegováním na objekt Strategy.

```
void Update()
{
    behaviour.DoBehaviour();
}
```

Obrázek 7 Vzor Strategy eliminuje podmínky

Tento návrhový vzor je vhodné používat v situacích, kdy jeho využití sníží množství podmínek a zpřehlední kód.

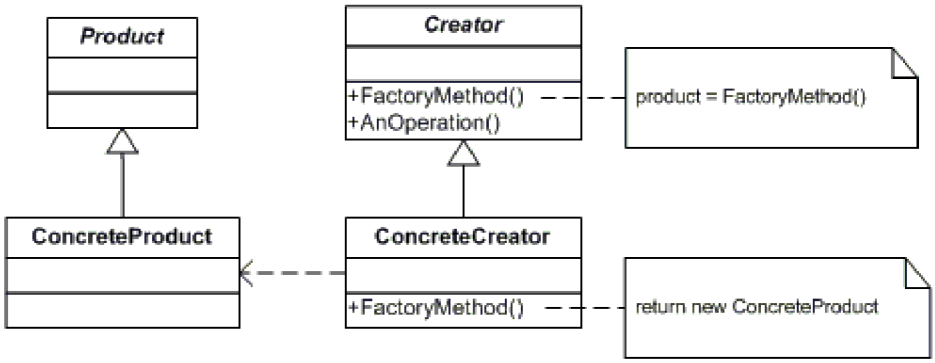
Využívání tohoto vzoru může vést k zvýšenému množství objektů. Jelikož každá instance objektu využívající tento vzor bude vytvářet další objekty pro jednotlivé algoritmy. Této vlastnosti návrhového vzoru se lze částečně vyhnout definováním statických podtříd třídy Strategy. S tímto přístupem je sdílená jedna instance každé strategie více objekty a je sníženo množství využívané paměti. Na tomto principu funguje také návrhový vzor Flyweight. (Gamma, 1995); (Kerievsky, 2004)

4.4 Factory

4.4.1 Účel vzoru

Slouží k vytvoření jednotného způsobu vytváření nových objektů a delegování jejich vytvoření konkrétním podtřídám. Vzor umožňuje abstrahovat vytváření nových objektů z pohledu ostatních tříd. To zajistí jednotný způsob vytváření nových objektů a předchází chybám, které mohou nastat v případech, kde je potřeba vytvářet větší množství objektů. (Tutorials Point, 2022)

4.4.2 Diagram



Obrázek 8 Diagram tříd návrhového vzoru Factory (Gamma, 1995)

- Třída Creator – Představuje obecný interface pro vytváření objektů. Objekty mohou používat tuto třídu, aniž by byly vázány na konkrétní vytvářené objekty.
- Třída ConcreteCreator – Implementuje metody nadtřídy Creator pro konkrétní vytvářený objekt.
- Třída Product – Představuje obecný objekt, který třída Creator vytváří.
- Třída ConcreteProduct – Představuje konkrétní vytvářenou třídu.

4.4.3 Výhody a nevýhody

Vhodnost využití vzoru Factory závisí na velikosti a složitosti projektu. Problém, který řeší, nenastává v situacích, kde je potřeba vytvořit jednotky objektu a jeho využití může naopak zbytečně zvýšit komplexitu aplikace.

Výhodou vzoru Factory je naopak snížení komplexity a zvýšení přehlednosti kódu v situacích, kde je vytvářeno větší množství objektů různými aplikačními systémy. (Gamma, 1995)

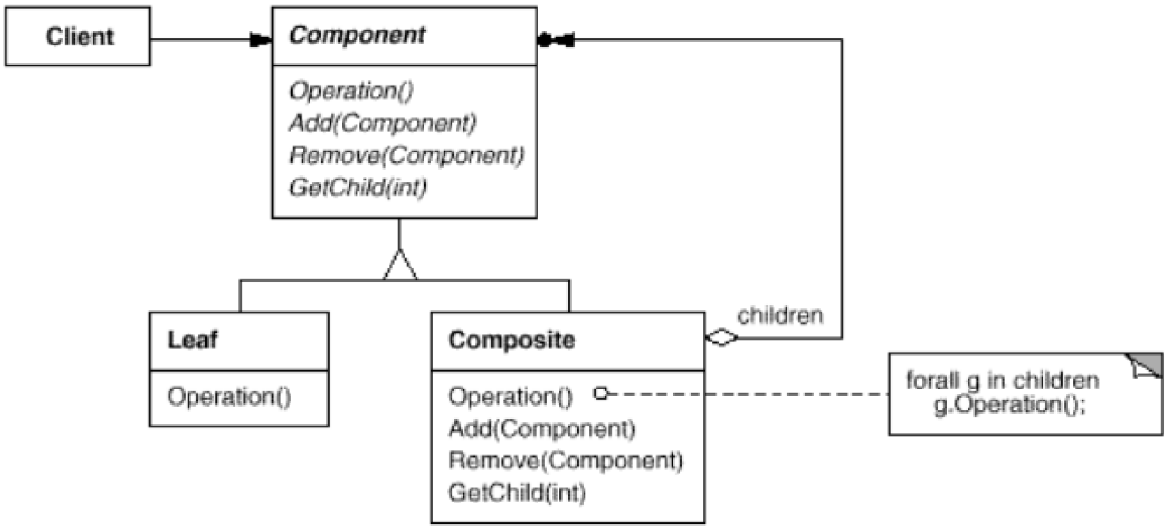
4.5 Composite

4.5.1 Účel vzoru

Vzor Composite umožňuje skládání objektů do hierarchických struktur a umožňuje jednotný přístup k jednotlivým objektům díky společnému rozhraní.

Tento vzor je využívám herními enginy ke skládání a upravování herních objektů. To umožňuje modulárně měnit funkcionalitu objektů změnou objektů v hierarchii a zároveň udržet nezávislost jednotlivých domén projektu. (Nystrom, 2014)

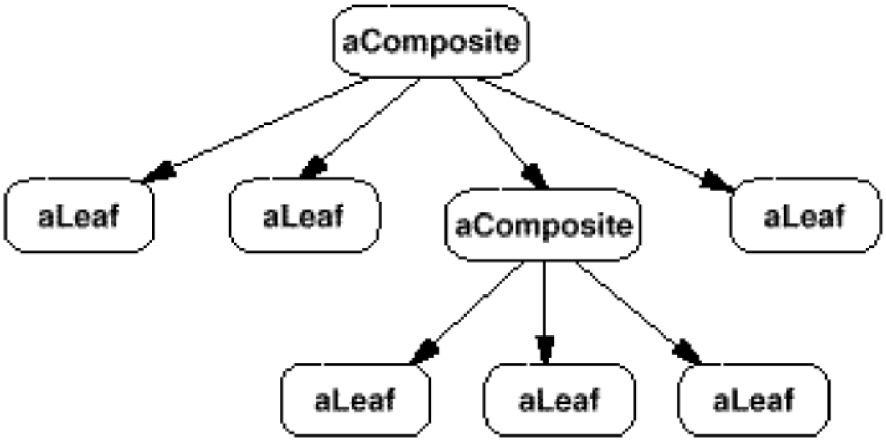
4.5.2 Diagram



Obrázek 9 Diagram tříd návrhového vzoru Composite (Gueheneuc, 2001)

- Třída Component – Nadtřída pro všechny objekty v hierarchii.
- Třída Leaf – Představuje konečný uzel hierarchie. Neobsahuje další objekty třídy Component.
- Třída Composite – Představuje objekt, který obsahuje další objekty třídy Component. Metodou Add lze vložit odkaz na další objekt typu Component a tímto dosáhneme ‚skládání' výsledného objektu.

Struktura rozložení objektů v hierarchii může vypadat takto:



Obrázek 10 Struktura hierarchie objektů využívající vzor Composite (Gamma, 1995)

4.5.3 Využití v herních enginech

Návrhový vzor Composite je využíván v editorech, kde lze k jednotlivým objektům modulárně přidávat další vlastnosti a také je odebírat. V herním engine Unity je tento vzor využíván pro práci s objekty třídy GameObject. GameObject odpovídá v diagramu třídě Composite. Jejich funkcionalitu lze následně rozšířit objekty Component. Ty jsou následně odpovědné za jednotlivé vlastnosti objektu, jako audio, renderování grafiky a uživatelem definované skripty. Objekt typu GameObject může následně hierarchicky obsahovat další GameObject a může být vytvořena komplexní struktura. (Lengyel, 2011)

Implementace v Unity by mohla vypadat takto:

```
public abstract class Object
{
    public virtual void Update();
}

public abstract class GameObject: Object
{
    private List<Object> children;

    public void AddChild(Object object);
    public void RemoveChild(Object object);
    public GameObject GetChild(int i);
}

public abstract class Component: Object
{

}
```

Obrázek 11 Zjednodušená implementace vzoru Composite v Unity

Funkce nejsou vypsány pro zkrácení ukázky.

4.5.4 Výhody a nevýhody

Výhodou vzoru Composite je modularita a možnost rozšiřování hierarchie objektů. Umožňuje sestavovat komplexní struktury a dynamicky je měnit za běhu programu. Procházení této hierarchie může být komplikovanější. Usnadnit průchod hierarchii lze udržováním odkazu každého objektu na jeho rodiče a řazením dětí.

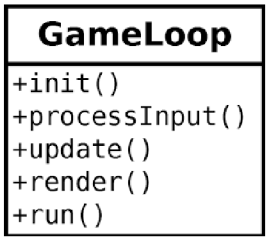
Další problém může způsobit nutnost rozlišení objektů Leaf a Composite při přidávání a odebírání vnořených objektů. S objekty typu Component nelze pracovat stejně, protože nemá metody Add a Remove, které má třída Composite. Můžeme to vyřešit přidáním metod i do třídy Leaf, které budou prázdné.

4.6 Game loop

4.6.1 Účel vzoru

Vzor Game loop, stejně jako další diskutované vzory, je specifický pro jeho využití v herních aplikacích. Ve hrách je nutné reagovat na vstupy hráče a vykreslovat změny na obrazovce nezávisle na probíhajících výpočtech programu. To umožňuje právě vzor Game loop.

4.6.2 Diagram



Obrázek 12 Diagram třídy vzoru Game Loop (Gosselin, 2022)

Ve hrách je zapotřebí opakovaně zpracovávat vstup hráče, pohyby objektů a vykreslovat je. Cyklus Game Loop běží po celou dobu hry. V každém kroku cyklu je zpracován vstup, aktualizován stav hry a vykreslena grafika. (Nystrom, 2014); (Madhav, 2013); (Gosselin, 2022)

```
while (true)
{
    ProcessInput();
    Update();
    Render();
}
```

Obrázek 13 Jednoduchá implementace vzoru Game Loop

Počet opakování tohoto cyklu je označován jako počet snímku za sekundu (FPS – z angličtiny frames per second). Příliš vysoké množství snímků za sekundu může být zbytečně náročné a může například zvyšovat spotřebu energie u mobilních zařízení. Malé množství snímků naopak může způsobit nepřesné fyzikální výpočty. Řešením je oddělení aktualizace stavu do samostatného cyklu s pravidelnými intervaly, nezávislými na rychlosti vykreslování.

```
while (true)
{
    double current = GetCurrentTime();
    double elapsed = current - previous;
    previous = current;
    lag += elapsed;

    ProcessInput();

    while (lag >= MS_PER_UPDATE)
    {
        Update();
        lag -= MS_PER_UPDATE;
    }

    Render();
}
```

Obrázek 14 Vzor Game Loop s funkcí Update nezávislou na FPS (Nystrom, 2014)

Při této implementaci není volání funkce Render nějak omezeno, ale funkce Update je volána jen po uplynutí předem daného časového intervalu. Tím je zajištěno, že k aktualizaci stavu hry nedochází zbytečně často.

Další možností rozšíření tohoto vzoru je omezení maximálního množství snímků obrazovky nebo synchronizace se snímky obrazovky. (Nystrom, 2014); (Madhav, 2013)

4.6.3 Výhody a nevýhody

Tento vzor je ve svém využití odlišný od ostatních. S výjimkou starých herních aplikací neexistuje aplikace, která by jej nevyužívala. Je součástí všech herní enginů. Spolu se vzorem Composite a Update tvoří jejich základ. Kód, který je psán uživateli herních enginů, je volán metodami tohoto vzoru.

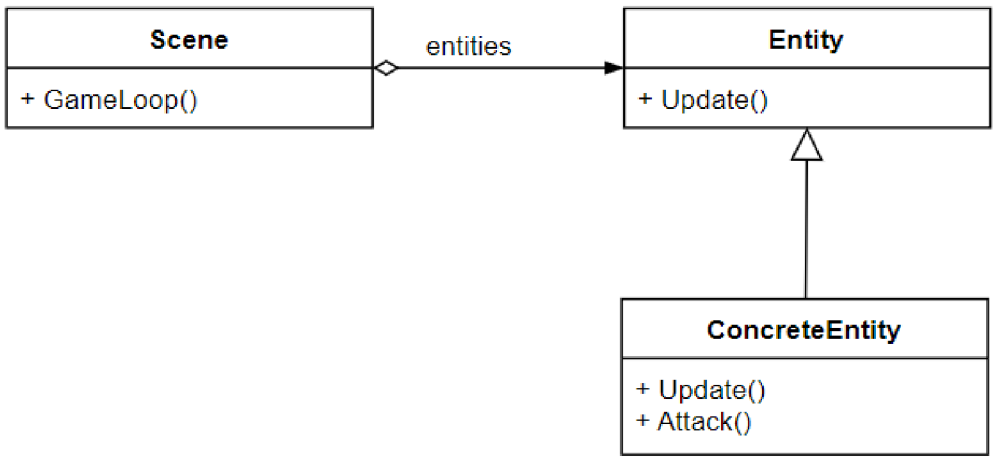
Velká část času běhu programu je strávena v tomto cyklu. Je proto důležité, aby kód volaný metodami toho vzoru, byl co nejefektivnější a nevyužíval příliš mnoho zdrojů.

4.7 Update

4.7.1 Účel vzoru

V herním světě je mnoho objektů, které je potřeba simulovat zároveň. Vzor Update umožňuje vytvořit iluzi plynulosti rozdělením chování na jednotlivé snímky. Tento návrhový vzor je využíván herními enginy a úzce souvisí se vzorem Game loop. Vzor Update rozšiřuje vzor Game Loop o nezávislost entit v herním světě a možnost jejich vznikání a zanikání. (Chamillard, 2022)

4.7.2 Diagram



Obrázek 15 Diagram tříd využívajících metodu Update

Diagram ukazuje příklad využití metody update v herní aplikaci. Třída Scene obsahuje seznam všech tříd Entity. Představuje aktuální scénu hry. Třída Entity představuje jednotlivé postavy nebo herní objekty. Metoda GameLoop je funkce podle vzoru Game Loop, ve které je cyklus volající metodu Update. (Nystrom, 2014)

4.7.3 Výhody a nevýhody

Podobně jako u Game Loop, je toto univerzálně využívaný vzor. Úzce spolu souvisí, a proto jsou jejich výhody a nevýhody podobné.

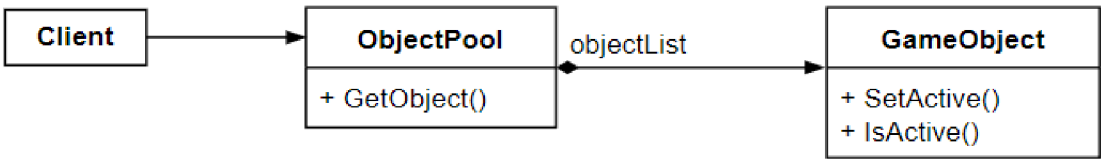
4.8 Object pool

4.8.1 Účel vzoru

Při vytváření a zanikání velkého množství objektů dochází k nutnosti opakovaně alokovat a uvolňovat paměť. Práci s pamětí lze zefektivnit znovupoužitím instancí objektů pomocí vzoru Object pool.

Object pool je objekt, který obstarává vytváření nových objektů a také jejich zánik. Ukládá odkaz, na již nevyužívané objekty, které v případě potřeby znovu obnoví pro další použití. (Nystrom, 2014); (Kowalski, 2018)

4.8.2 Diagram



Obrázek 16 Diagram tříd vzoru Object pool

- Třída ObjectPool – Třída obsahuje pole všech vytvořených objektů, tedy samotný ‚pool', v kterém jsou uloženy aktivní a neaktivní objekty. Metoda GetObject vrací odkaz na objekt třídy GameObject který není aktivní a označí ho jako aktivní. Pokud žádný takový objekt v poli objectList není, je vytvořen nový.
- Třída GameObject – Třída představuje konkrétní objekty, které vzor obstarává. Objekt si uchovává stav, podle kterého je považován za používaný nebo ne. Metoda SetActive slouží k aktivování tohoto objektu. Metoda IsActive vrací aktuální stav.

```
public GameObject GetObject()
{
    for (int i = 0; i < pool.Count; i++)
    {
        if (pool[i].IsActive() == false)
        {
            pool[i].SetActive(true);
            return pool[i];
        }
    }

    GameObject pooledObject = new GameObject();
    pooledObject.SetActive(true);
    pool.Add(pooledObject);
    return pooledObject;
}
```

Obrázek 17 Implementace metody GetObject

V základní implementaci metoda GetObject prochází pole všech objektů a pokud najde neaktivní, aktivuje jej a vrátí jako návratovou hodnotu funkce. Pokud takový objekt nenajde, vytvoří nový.

Pole objectList je možné nahradit dvěma poli. Jedno pro aktivní a jedno pro neaktivní objekty. To přinese určité vlastnosti algoritmu. Objekty třídy GameObject nebudou muset ukládat svůj stav a zjednoduší se náročnost vyhledávání neaktivních objektů v metodě GetObject. Ovšem pokud je stav třídy GameObject důležitý pro její ostatní metody, budou si ho muset ukládat i na dále a odstranění vyhledávání v poli přinese znatelnou výhodu jen při vysokém množství objektů.

Pokud je znám předem maximální počet objektů, které budou využívány, je možné algoritmus upravit a objekty GameObject vytvořit při vytváření objektu ObjectPool. Objekty jsou tak vytvořeny v předem známý čas na začátku běhu programu a předejde se pak jejich vytváření za běhu. Pokud je jejich vytváření náročné na zdroje, může být toto správná volba. Ovšem pokud nejsou využívány všechny objekty, program pak zablokuje více paměti, než skutečně využívá. (Nystrom, 2014)

4.8.3 Výhody a nevýhody

Tento vzor využívá jeden ze základních způsobů, jak se vyhnout nepředpověditelné zátěži, kterou v některých jazycích způsobuje systém uvolňování paměti (anglicky Garbage collection). Tím, že se objekty recyklují, není potřeba uvolňovat paměť. Výhoda využití tohoto vzoru je v tomto ohledu jednoznačná. Ovšem má také určitá omezení. V situaci, kdy jsou objekty v ‘poolu‘ využívány jen po část běhu programu, zůstanou v paměti po celou dobu od

jejich vytvoření. To může vést k zbytečně zvýšenému využití paměti v situaci, kdy ji aplikace nevyužívá. Takové situace je potřeba ošetřit a zohlednit je při návrhu.

Využití tohoto vzoru vede k větší komplexitě projektu. Konkrétní přínos ušetření volání paměti může mít velkou hodnotu v určitých projektech, ale také může zbytečně zvýšit složitost kódu a snížit jeho přehlednost. (Kowalski, 2018)

4.9 Vývoj v herním engine Unity

Unity je velice komplexní a rozsáhlý herní engine. Tato kapitola je krátkým úvodem pro práci v Unity a referencí pro pojmy používané v následujících kapitolách.

4.9.1 Třídy GameObject a Component

Základním stavebním kamenem Unity je třída GameObject. Všechny objekty, se kterými hráč interaguje a které jsou vykreslovány na obrazovku nebo přehrávají audio, jsou třídy GameObject. Objekty třídy GameObject musí být vždy vytvořeny v kontextu aktuální scény. Scéna představuje aktuální úroveň, menu nebo celé prostředí kde se hra odehrává.

GameObject může obsahovat další hierarchicky vnořené GameObjecty a libovolné množství objektů třídy Component. Component představuje přidanou funkcionalitu k samotnému GameObjectu. Unity obsahuje desítky vestavěných Component. Mezi základní patří komponenty SpriteRenderer a MeshRenderer pro renderování 2D a 3D grafiky, AudioSource a AudioListener přehrávání audio klipů a Animator pro přehrávání animací. Komponenty lze rozšiřovat a vytvářet vlastní. Při skriptování vlastních komponent je nutné dědit z třídy MonoBehaviour.

Každý GameObject vždy obsahuje komponentu Transform. Ta obsahuje informace o pozici a rotaci objektu ve scéně.

4.9.2 Vytváření vlastních komponent třídy MonoBehaviour

Objekty dědící z třídy Monobehaviour lze vložit do objektů GameObject ve scéně. To umožňuje vytvářet vlastní komponenty využívající funkce Unity.

Vytváření nových instancí objektů MonoBehaviour a pořadí, ve kterém jsou vytvořeny, určuje samotný engine. Není proto možné využívat konstruktory. Namísto toho jsou k dispozici funkce, které engine volá v určitých situacích.

Základní funkce jsou Awake a Start, které se volají po vytvoření objektu, před vykreslením prvního snímku. Ty se využívají pro inicializaci objektu a referencí.

Funkce Update a FixedUpdate jsou funkce návrhového vzoru Update. Update je volán při vykreslení každého snímku a FixedUpdate po uplynutí daného času, výchozí frekvence je jednou za 0.02 sekund. To je využíváno pro přesné fyzikální výpočty.

Monobehaviour obsahuje dalších funkce pro fyzikální interakce a práci s game objekty.

4.9.3 Komponenta Collider

Komponenty Collider definují tvar objektu GameObject pro účely fyzikálních kolizí. Collider, který je neviditelný, nemusí mít přesně stejný tvar jako model nebo obrázek objektu GameObject. Hrubá aproximace modelu je často efektivnější a při hře k nerozeznání. (Unity User Manual, 2022)

Komponenta Collider lze také využít pro detekci okamžiku, kdy jeden objekt s komponentou Collider vstoupí do prostoru druhého, aniž by vznikla kolize. Takovéto collidery se označuji jako Trigger.

4.9.4 Pojmy

Pro pochopení implementace návrhových vzorů popsané v následují kapitole je potřeba se orientovat pojmech specifických pro Unity. Zde je krátký rejstřík základních pojmů:

- MonoBehaviour – Základní třída, ze které dědí všechny komponenty.
- Komponenta – Objekt, který lze připojit k objektům třídy GameObject.
- GameObject – Základní entita v Unity scéně.
- Scéna – Obsahuje všechny aktivní herní objekty.
- Collider – Komponenta umožňující fyzikální interakce.

5 Herní aplikace

5.1 Návrh

5.1.1 Cíl

Při návrhu herní aplikace bylo vycházeno ze seznamu návrhových vzorů zmíněných v této práci. Při vývoji této aplikace budou využity i další návrhové vzory, které budou zmíněny v poslední kapitole.

Cílem aplikace je využití návrhových vzorů v herní aplikaci a demonstrování jejich vlastností.

V následující části bude vysvětlena hlavní myšlenka hry. Poté budou rozebrána konkrétní využití návrhových vzorů v objektovém návrhu.

5.1.2 Návrh

V této herní aplikaci hráč ovládá postavu potápěče, který se opakovaně potápí pod hladinu a loví podmořské ryby. Odměnou za ulovené ryby jsou peníze, za které si může koupit vylepšení, které mu umožní potápět se hlouběji. Cílem hry je potopit se na mořské dno a vylovit poklad. Překážkou v dosažení cíle jsou nebezpečné ryby a omezené množství kyslíku. Pokud hráč umře, musí začít od začátku.

Hra se ovládá klávesnicí a myší. Prostředí herního světa je dvoudimenzionální v pohledu ze strany. Pohled hráče se posouvá vertikálně podle jeho pozice.

Obrázek 18 Herní obrazovka s postavou hráče

5.1.3 Herní prvky

V této kapitole jsou představeny základní herní prvky, se kterými hráč interaguje.

5.1.3.1 Postava a ovládání

Pohyb postavy se ovládá pomocí šipek na klávesnici nebo kláves W, A, S a D. Pohyb využívá fyzikálního modelu herního enginu k simulaci gravitace a setrvačnosti. Po odemknutí vylepšení může hráč skákat pomocí klávesy Shift. To umožňuje zrychlený pohyb pod vodou.

Hráč má určitý počet životů, o které když přijde, hra končí. Životy klesají, pokud se hráč dotkne nebezpečné ryby nebo pokud klesne počítadlo kyslíku na nulu.

Počítadlo kyslíku ukazuje, kolik času může hráč strávit pod vodou. Čas lze zvýšit pomocí vylepšení.

5.1.3.2 Ryby

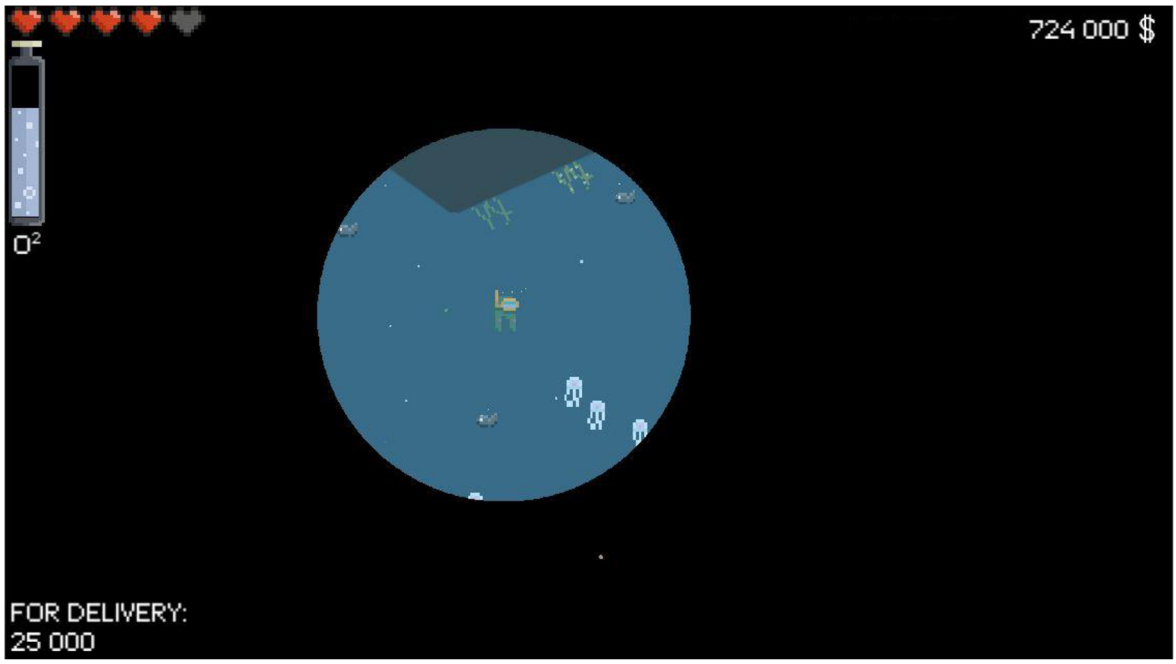
Hlavním cílem hry je sbírání ryb. Ryby se průběžně objevují v herním světě a nahodile se v něm pohybují. Druh ryb je rozdílný podle hloubky, ve které se vytvoří. Některé ryby jsou nebezpečné pro hráče a při kontaktu s nimi klesnou hráči životy.

5.1.3.3 Vylepšení

Při každém vynoření má hráč možnost utratit našetřené peníze za vylepšení. Mezi vylepšení patří: Zvýšení počítadla kyslíku, zvýšení počtu životů, světlo, skok a doplnění chybějících životů.

5.1.3.4 Poklad

Na spodním konci herního světa se nachází jeskynní útvar, ve kterém je schovaný poklad. Cílem hry je tento poklad najít a vynést na hladinu.



Obrázek 19 Hráč hledá poklad v hlubinné jeskyni

5.2 Využití návrhových vzorů

Tato herní aplikace je vytvořena v herním engine Unity za využití programovacího jazyku C#. Návrhové vzory jsou upraveny tak, aby objekty využívali zabudovaných funkcionalit a vlastností herního engine. To vede například k tomu, že objekty, které existují v kontextu scény, jsou vytvářeny herním enginem a není proto možné používat konstruktor. Ten lze ale nahradit funkcemi Awake nebo Start, které jsou vždy zavolány po vytvoření objektu.

5.2.1 Singleton

Návrhový vzor Singleton byl využit v této aplikaci vícekrát. Jeho využití je ilustrováno na třídě pro zprávu uživatelského rozhraní. Třída UserInterface má na starost umožnit zobrazení

údajů na grafických elementech uživatelského rozhraní. Tyto elementy představují životy, zbývající množství kyslíku a peníze. Objekty, které upravují zobrazované hodnoty, potřebují referenci na instanci třídy UserInterface. Zároveň vždy musí existovat právě jedna instance. Tyto vlastnosti zajišťuje vzor Singleton.

Implementace vzoru je pozměněná od diagramu v teoretické části této práce. Vzhledem k tomu, že je žádané, aby objekt byl součástí scény a obsahoval odkazy na objekty ve scéně, je třída komponentou, tedy dědí z třídy MonoBehaviour. Kvůli tomu není možné pro nastavení reference a zajištění jediné instance využití konstruktoru. Namísto toho je instance nastavena v metodě Awake.

```
private static UserInterface _instance;

public static UserInterface Instance { get { return _instance; } }

//Awake se zavolá po vytvoření třídy a nastaví odkaz _instance
private void Awake()
{
    if (_instance != null && _instance != this)
    {
        Destroy(this);
    }
    else
    {
        _instance = this;
    }
}
```

Obrázek 20 Implementace vzory Singleton v třídě UserInteface

Metoda Awake je zavolána hned po vytvoření objektu ve scéně. Podmínka v této metodě zajišťuje jedinečnost této instance. Pokud je vytvořeno více instancí, je aktuální instance odstraněna pomocí metody Destroy s argumentem reference na tuto třídu.

Proměnná _instance je statická, a pokud neobsahuje referenci je nastavena na aktuální instanci. Ostatní objekty přistupují k referenci pomocí veřejné proměnné Instance, které vrátí hodnotu proměnné _instance.

5.2.2 Observer

Návrhový vzor Observer je pro mnoho případů jednostranné komunikace mezi objekty. Tento vzor lze v Unity implementovat více způsoby. Zde jsou jmenovány tři hlavní.

- Vlastní implementací
- Pomocí C# delegátů
- Využitím třídy UnityEvent

První možností je implementovat abstraktní třídy pro pozorovatele (Observer) i subjekt (Subject) s logikou metod pro přihlášení, odhlášení a aktualizaci. Tento přístup umožňuje nejvíce upravit chování objektů.

Součástí programovacího jazyka C# jsou delegáti, kteří nám umožňují pracovat s referencemi na metody. To lze ve vzoru Observer využít a subjekt může obsahovat pouze pole metod. Díky tomu není nutné využívat dědění a umožníme přihlášení k Subjekt všem metodám splňujícím signaturu delegátu. Delegáty také mohou obsahovat argumenty, přes které je možné předávat stav subjektu a vyhnout se nutnosti zpětného dotazu od pozorovatele. Argumenty lze využít i při vlastní implementaci.

Třetí přístup, který byl použit v projektu, přenechává implementaci na Unity. Využívá třídu UnityEvent, která je součásti knihovny UnityEngine.

Umožňuje graficky pracovat s polem propojených objektů v inspektoru. Díky tomu s ním lze snadno pracovat. To se může být vhodné především, pokud pracujete v týmu s lidmi, kteří nepracují přímo s kódem.

```
namespace UnityEngine.Events
{
    public class UnityEvent : UnityEventBase
    {
        [RequiredByNativeCode]
        public UnityEvent();
        public void AddListener(UnityAction call);
        public void Invoke();
        public void RemoveListener(UnityAction call);
        protected override MethodInfo FindMethod_Impl(string name, Type targetObjType);
    }
}
```

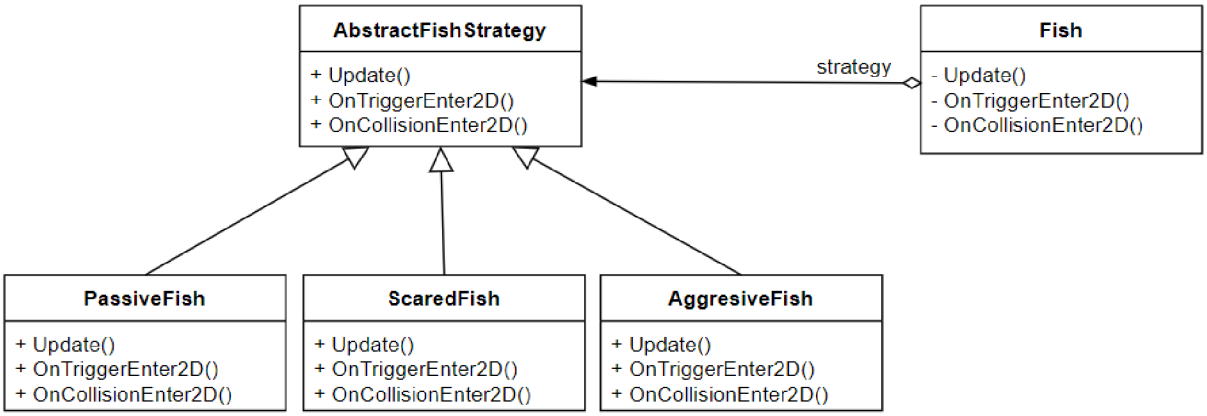
Obrázek 21 Metody objektu UnityEvent

Třída UnityEvent obsahuje metody AddListener a RemoveListener pro přidání a odebrání volaných metod. UnityAction je C# delegát. Metoda Invoke slouží k vyvolání volání všech registrovaných metod.

Práce s třídou UnityEvent je obdobná klasické implementaci vzoru Observer, ale propojenost s editorem umožňuje přehlednější práci s volanými metodami.

5.2.3 Strategy

Návrhový vzor Strategy je využit v aplikaci pro různé druhy chování ryb. Jednotlivé ryby se odlišují tím, jak reagují na hráče a také tím, jestli hráči mohou ublížit.



Obrázek 22 Diagram tříd využívajících vzor Strategy

Třída AbstractFishStrategy je abstraktní třída, která implementuje společné metody a proměnné stavů. Třídy AggresiveFish, PassiveFish, ScaredFish a DangerousFish představují jednotlivé způsoby chování ryb. Ryba se strategií AggresiveFish zaútočí na hráče, pokud se k ní přiblíží a při kontaktu mu ubírá životy. Ryba, která je pouze nebezpečná a využívá strategii DangerousFish, nebude hráče pronásledovat, ale pokud se jí hráč dotkne, ubere mu životy. Strategie ScaredFish je opakem AggresiveFish. Takovéto ryby před hráčem utíkají a neubírají mu životy. PassiveFish je nejjednodušší z typů chování. Nijak nereaguje na hráče a neubírá mu životy.

Objekty vzoru Strategy implementují tři veřejné metody: Update, OnCollisionEnter, OnTriggerEnter. Tyto metody jsou volány z třídy Fish, která představuje samotnou rybu. Metoda Update je metoda podle vzoru update, obstarává pohyb ryb a liší se v jednotlivých implementacích schopností ryb změnit rychlost, pokud pronásledují hráče nebo před ním utíkají. Metoda OnCollisionEnter je volána při kontaktu ryby s hráčem a implementuje schopnost ubírat hráči životy. Metoda OnTriggerEnter je volána, pokud hráč vstoupí do trigger collideru, který na sobě mají ryby. Tento typ collideru není pevný a ostatní fyzické collidery s ním nekolidují a pouze vyvolají metodu OnTriggerEnter. To umožňuje jednoduchou detekci blízkosti dvou objektů.

5.2.4 Object Pool

Návrhový vzor Object Pool je vhodné využít pro situaci, kde je vytvářeno a rušeno větší množství objektů. V popisované aplikaci toto platí pro objekty představující ryby. V herní scéně jsou stále vyvářeny nové ryby a hráč je stále sbírá.

Třída FishObjectPool využívá tento vzor a zároveň nastaví rybám výchozí strategii, podle vzoru Strategy. Třída také dědí z třídy ScriptableObject. to umožňuje práci s instancemi této třídy nezávisle na scéně.

K vytváření objektů ryb jsou využity tzv. prefaby. Ty představují Game Object, který je uložený v paměti a při vytváření ryb. Vytvoříme jeho kopii, kterou vložíme do scény. Toto je běžný přístup v Unity k vytváření nových objektů, které jsou předem připravené v editoru.

Třída FishObjectPool je rozšířena o možnost nastavení výchozího počtu ryb. Po spuštění hry je na zavolána funkce OnEnable. V této funkci jsou vytvořeny ryby soukromou metodou CreateFish a přidány do pole s dostupnými objekty. Ryby jsou vytvářeny neaktivní pomocí metody SetActive v třídě Game Object. Pokud je touto metodou nastaven Game Object na neaktivní, nejsou volány metody jako Awake, Start, Update, FixedUpdate. Objekt se také neúčastní fyzikální simulace a není vykreslován.

Přístup k rybám si ostatní objekty vyžádají pomocí metody GetAvailableObject, který jim vrátí odkaz na aktivovanou rybu.

```
public GameObject GetAvailableObject()
{
    for (int i = 0; i < pool.Count; i++)
    {
        if (pool[i].activeInHierarchy == false)
            return pool[i];
    }

    if (canGrow == true)
    {
        return CreateFish();
    }
    else
        return null;
}
```

Obrázek 23 Metoda GetAvailableObject

Metoda GetAvailableObject prochází pole vytvořených objektů a hledá neaktivní objekt, který není využíván. Pokud ho najde, vrátí odkaz na něj jako návratovou hodnotu. V případě, že žádný takový objekt neexistuje, dojde ke kontrole, zda je dovoleno vytvářet nové objekty. Pokud ano, je zavolána metoda CreateFish pro vytvoření nové ryby a vrácen odkaz na ni.

Využití třídy Scriptable Object umožňuje práci s instancemi objektů v editoru, aniž by byly pevně vázané na GameObject. Z pohledu využití mají některé podobné vlastnosti jako statické objekty, ale mohou mít více instancí. (Unity User Manual, 2022)

5.2.5 Composite, Game Loop a Update

Projekt využívá implementaci návrhových vzorů Composite, Game Loop a Update v herním engine Unity.

V Unity jsou všechny objekty ve scéně typu GameObject. Tento objekt sám o sobě nemá žádnou funkci, ale lze jej rozšířit přidáním Component. Do každého game objektu lze také hierarchicky vkládat další objekty. Toto je princip vzoru Composite. Je využíván prakticky ve všech herních enginech, jelikož umožňuje oddělení a snadnou práci s jednotlivými moduly, jako je například audio, renderování a logika objektů.

Využití vzoru Game Loop se nedá vyhnout u prakticky všech grafických her a využívají ho i některé neherní aplikace. V unity je implementace tohoto vzoru velice komplexní a iterace samotného cyklu obsahuje mnoho kroků.

Vzor Update je v Unity reprezentován dvěma metodami. Metoda FixedUpdate je volána po předem daných časových intervalech. Výchozí interval je 50 volání za sekundu. Je určen především pro výpočet fyzikálních interakcí. Pravidelný interval zajistí stejné chování programu na různých strojích. V Unity jsou objekty, na které působí fyzikální interakce simulovány pomocí komponenty Rigidbody. Ta slouží k simulaci gravitace, vlivu sil a detekci kolizí. Operace tohoto komponentu běží v metodě FixedUpdate. Rygidbody vyžívám v aplikaci pro simulaci pohybu ryb a hráče.

Metoda Update je druhou metodou vzoru Update. Je volána při vykreslování každého snímku. Frekvence volání je závislá na náročnosti aplikace a výkonu počítače. V mé aplikaci slouží k detekci vstupů pro pohyb hráče. Přehrávání některých zvukových efektů a animací. Vyšší frekvence volání metody Update umožňuje okamžité vizuální reakce a celá hra je plynulejší než za využití pouze metody FixedUpdate.

5.3 Shrnutí

Herní aplikace byla vytvořena za využití návrhových vzorů probíraných teoretické části. Návrhové vzory Singleton, Observer, Strategy a Object Pool byly implementovány a využity v herním systému vyvíjené aplikace. V Některých případech využívala jedna třída dva z těchto návrhových vzorů, konkrétně Singleton a Observer.

Návrhový vzor Factory nebyl v práci využit, protože v herní aplikaci nebyl vhodný případ pro jeho použití.

Většina objektů v herní aplikaci využívá implementaci návrhových vzorů Game Loop, Composite a Update v Unity. Využívány jsou metody těchto vzorů, které jsou volány herním enginem.

Využití návrhových vzorů Singleton a Observer vedlo k jednodušší práci s referencemi na instance jednotlivých objektů. Vzor Strategy umožnil zpřehlednit třídu Fish, která by jinak obsahovala daleko více řádek kódu. Vzhledem k velikosti projektu by toto nevedlo k velkým problémům, ale byla tím demonstrována silná stránka tohoto návrhového vzoru.

Podobně jako u vzoru Observer, využití vzoru Object pool nepřineslo výrazné výhody oproti alternativě. Objekty ryb zabírají malé množství paměti a dopad na výkon nebyl z pohledu hráče znatelný. Pokud by ovšem v budoucí verzi aplikace byla herní oblast rozšířena a množství entit by se několikanásobně zvýšilo, rozdíl mezi implementací využívající vzor Object pool a alternativu by se prohloubil.

Herní aplikace využila a demonstrovala některé návrhové vzory, využívané v herním vývoji.

6 Závěr

Návrhové vzory nejsou pro mnoho vývojářů oblíbeným tématem. Jedním z cílů této práce bylo poukázat na jejich silné stránky, a proč je důležité návrhovým vzorům porozumět.

Návrhové vzory jsou návody na řešení problémů, se kterými se vývojáři běžné setkávají. Porozumění procesům, které byly již ověřeny a testovány, povede ke zrychlení vývoje. Tato práce představila seznam několika základních vzorů, se kterými by se měli seznámit začínající vývojáři v oblasti herního vývoje. Zapojení těchto vzorů do návrhu aplikací vede k lepší struktuře kódu, k optimalizaci využití paměti a k jednoduššímu vytváření nových objektů. Tento seznam byl vytvořen a konkrétní návrhové vzory byly představeny v jednotlivých kapitolách. Představena byla jejich implementace pomocí UML diagramů tříd. Také byly uvedeny jejich vlastnosti, výhody, nevýhody a případy použití.

Dílčím cílem bylo vytvoření herní aplikace, která ukázala využití návrhových vzorů prezentovaných v teoretické části. Některé byly přímo implementovány za použití skriptu v jazyce C# a u některých byla využita jejich implementace přímo v herním engine Unity. Herní aplikace byla nejdříve navrhnuta a následně byla podle návrhu vytvořena. Výsledná aplikace demonstrovala přínos využití některých návrhových vzorů a také jejich některé nevýhody v projektu malého rozměru.

Tato práce vzhledem k danému rozsahu nepokrývá celou problematiku návrhových vzorů a hernímu vývoji se věnuje jen zkráceně. Vytvořila ale určitý průřez těchto problematik, který může sloužit jako základ pro programátory, kteří se chtějí vzdělávat ve vývoji herních aplikací.

Seznam použitých zdrojů

ALEXANDER, Christopher, Sara ISHIKAWA, Murray SILVERSTEIN, Max JACOBSON, Ingrid KING a Shlomo ANGEL, 1977. *A pattern language: towns, buildings, construction*. 1. New York: Oxford University Press. ISBN 978-0-19-501919-3.

AMPATZOGLOU, Apostolos a Alexander CHATZIGEORGIOU, 2007. Evaluation of object-oriented design patterns in game development. *Information and Software Technology*. **49**(5), 445-454. ISSN 09505849. Dostupné z: doi:10.1016/j.infsof.2006.07.003

BISHOP, L., D. EBERLY, T. WHITTED, M. FINCH a M. SHANTZ, 1998. Designing a PC game engine. *IEEE Computer Graphics and Applications*. **18**(1), 46-53. ISSN 02721716. Dostupné z: doi:10.1109/38.637270

FREEMAN, Eric, Elisabeth ROBSON, Kathy SIERRA a Bert BATES, 2014. *Head First design patterns*. 1st Edition. Beijing: O'Reilly. ISBN 978-0-596-00712-6.

GAMMA, Erich, Richard HELM, Ralph JOHNSON a John VLISSIDES, 1995. *Design patterns: elements of reusable object-oriented software*. 1st Edition. Boston: Addison-Wesley. Addison-Wesley professional computing series. ISBN 978-0201633610.

GOSSELIN, Philippe-Henri, 2022. Discover Python and Patterns (8): Game Loop pattern. In: *Patternsgameprog.com* [online]. Paříž: patternsgameprog.com [cit. 2022-02-19]. Dostupné z: https://www.patternsgameprog.com/discover-python-and-patterns-8-game-loop-pattern

GRUDL, David, 2008. Je singleton zlo?. In: *PhpFashion* [online]. Praha: phpFashion [cit. 2022-02-16]. Dostupné z: https://phpfashion.com/je-singleton-zlo

GUEHENEUC, Y.-G. a H. ALBIN-AMIOT, 2001. Using design patterns and constraints to automate the detection and correction of inter-class design defects. *Proceedings 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems. TOOLS 39*. IEEE Comput. Soc, 296-305. ISBN 0-7695-1251-8. Dostupné z: doi:10.1109/TOOLS.2001.941682

CHAMILLARD, Dr. Tim, 2022. Game Loop and Update Method. In: *Coursera* [online]. Kalifornie: Coursera Inc. [cit. 2022-02-19]. Dostupné z: https://www.coursera.org/lecture/data-structures-design-patterns/game-loop-and-update-method-RfT3p

KERIEVSKY, Joshua, 2004. *Refactoring to Patterns*. 1st Edition. Hoboken: Pearson Education. ISBN 9780321213358.

KEVURU GAMES, 2022. MOBILE GAMING VS PC GAMING: OVERVIEW AND PROSPECTS. In: *Kevuru Games* [online]. Delaware: Kevuru games [cit. 2022-02-16]. Dostupné z: https://kevurugames.com/blog/mobile-gaming-vs-pc-gaming-overview-and-prospects/

KOWALSKI, Sławomir, 2018. Design patterns: Object pool. In: *Medium* [online]. San Francisco: Medium [cit. 2022-02-19]. Dostupné z: https://medium.com/@sawomirkowalski/design-patterns-object-pool-e8269fd45e10

LENGYEL, Eric, ed., 2011. *Game Engine Gems 2*. 1st Edition. Boca Raton: CRC Press. ISBN 978-1568814377.

MADHAV, Sanjay, 2013. *Game Programming Algorithms and Techniques: A Platform-Agnostic Approach (Game Design)*. 1st Edition. Hoboken: Pearson Education. ISBN 0321940156.

NYSTROM, Robert, 2014. *Game Programming Patterns*. 1st Edition. Seattle: Genever Benning. ISBN 978-0990582908.

PECINOVSKÝ, Rudolf, 2007. *Návrhové vzory: [33 vzorových postupů pro objektové programování]*. Brno: Computer Press. ISBN 978-80-251-1582-4.

QASIM, Awais, Adeel MUNAWAR, Jawad HASSAN a Adnan KHALID, 2021. Evaluating the Impact of Design Pattern Usage on Energy Consumption of Applications for Mobile Platform. *Applied Computer Systems*. **26**(1), 1-11. ISSN 2255-8691. Dostupné z: doi:10.2478/acss-2021-0001

TUTORIALS POINT, 2022. Design Pattern - Factory Pattern. In: *Tutorials Point* [online]. Hyderabad: Tutorials Point [cit. 2022-02-16]. Dostupné z: https://www.tutorialspoint.com/design_pattern/factory_pattern.htm

Unity User Manual [online], 2022. San Francisco: Unity Technologies [cit. 2022-02-14]. Dostupné z: https://docs.unity3d.com/

W3SDESIGN, 2022. Observer. In: *W3sDesign* [online]. Graz: w3sDesign [cit. 2022-02-16]. Dostupné z: http://w3sdesign.com/?gr=b07&ugr=proble#gf