Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky
Katedra informačních technologií

Studijní program: Aplikovaná informatika
Obor: Informační systémy a technologie

Navrhněte počítačovou hru pro vstupní kurzy programování

DIPLOMOVÁ PRÁCE

Student : Bc. David Sedláček
Vedoucí : Ing. Rudolf Pecinovský, CSc.
Oponent : Ing. Jarmila Pavlíčková, Ph.D.

2018

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 3. prosince 2018 . . . . . .

David Sedláček

Poděkování

Zde bych chtěl poděkovat svému vedoucímu, panu Ing. Rudolfovi Pecinovskému, CSc., za vedení práce a podnětné rady a připomínky.

Abstrakt

Praktické ukázky ve cvičeních mají znatelný dopad na efektivitu výuky objektově orientovaného programování. Některé přístupy k výuce mohou studentům učení znesnadnit. Dobrým příkladem jsou případy cvičení, ve kterých se studenti příliš zabývají samotným kódováním. Dobře navržený přístup k výuce metodikou Architecture-first může zvýšit efektivitu učení a poskytnout větší přidanou hodnotu kurzu.

Práce zkoumá účinky různých přístupů návrhu cvičení a celých kurzů a navrhuje příklad výuky na základě postupně vylepšovaného projektu pro studeny vstupních kurzů programování s důrazem na návrhové vzory. Je třeba věnovat pozornost nejen přístupu k výuce ale také výukovým metodám, kontextu kurzu a jeho plánu.

Práce navrhuje přístup k výuce programování vstupních kurzů se snahou vyhnout se základním prvkům kódování a namísto toho nejprve představit řešení skrze ověřené návrhové vzory. Důležitost výuky návrhových vzorů spočívá v nastavení dobrých zásad pro budoucí softwarové inženýry a vývojáře.

Další výzkum na toto téma by se měl zaměřit na iterativní zlepšování návrhu cvičení na základě vyhodnocování kurzů a na zhodnocení stěžejních pedagogických vzorů, které by měly být v lekcích použity.

Klíčová slova

Vstupní kurzy programování, OOP (objektově orientované programování), Java, Architecture-first, výuka, programování, BlueJ, výuka vývojem hry, kontinuální vylepšování projektu, výuka návrhových vzorů, návrhové vzory, vývoj softwaru, vážná hra, gamifikace, výukové vzory

Abstract

Practical examples in exercises have significant impact on effectivity of teaching object-oriented programming. Some approaches may hinder student learning. Cases where students pay too much attention to coding itself instead of to high level design are good example of ineffective and possibly even harming teaching. A well-designed Architecture-first approach to teaching programming may improve student learning and value added by the course.

This study investigates the effects of different approaches to course and exercise design and designs example of continuously improved project for students of CS1 courses to learn basics of programming with emphasis on design patterns. Attention must be paid not only to approach to teaching, but to teaching techniques, context of use and course plan as well.

Study offers approach to teaching programming in introductory courses, which tries to avoid programming language specific tools if possible so that students first learn solid solution by design patterns. The importance of teaching design patterns lies in setting up a good practice for future software engineers and developers.

Future research is recommended to iteratively improve on exercise design based on student experience and to assess significant pedagogical patterns which should be used within lectures.

Keywords

CS1 (Computer Science 1), OOP (object-oriented programming), Architecture-first, Java, education, education via game development, software development, design patterns, continuous project improvement, design pattern education, serious game, gamification, pedagogical patterns

Obsah

1 Úvod

Práce se věnuje tématu výuky programování ve vstupních kurzech na vysokých školách, se zaměřením na objektově orientované programování a výuku návrhových vzorů.

1.1 Cíle práce

Cílem této práce je navrhnout postup výuky ve vstupních kurzech programování. Tohoto cíle bude dosaženo naplněním následujících bodů.

- Analýza námětů, které se ve vstupních kurzech programování používají jako základ postupně vylepšovaných programů
- Návrh vlastního námětu pro program, který by bylo možné použít při výuce podle metodiky Architecture First
- Návrh postupu výuky při postupném rozpracovávání výše navrženého námětu. Pro generování postupně zdokonalovaných projektů se využívá program Monolith2Projects

1.2 Cílová skupina

Práce je určena pro školitele a učitele programování, jejichž cílem je své studenty seznámit s objektově orientovaným programováním takovým způsobem, aby látku pochopili bez zbytečného noření se do problematiky konkrétního programovacího jazyka.

1.3 Použité metody

Používá se zde metoda systematické analýzy literatury, jejíž výstupy jsou obsaženy v kapitolách Práce je rozdělena na oddíly komentované rešerše, současného stavu problematiky, výběru námětu a navrhovaného postupu výuky. V teoretické části práce komentovaná rešerše dělí zdroje do tematických kapitol a shrnuje jejich přínos k problematice. Současný stav problematiky se věnuje v detailu skutečnostem, které ovlivňují návrh postupu výuky.

Vlastní práce obsahuje kapitoly věnující se výběru námětu, kde jsou jednotlivé náměty z teoretické části posuzovány. Na tyto kapitoly navazuje navrhovaný postup výuky, který je zde dělen do lekcí.

Komentovaná rešerše informačních zdrojů a Současný stav řešené problematiky. Vlastní práce je tvořena na základě metody Design Science Research.

1.4 Struktura práce

Práce je rozdělena na oddíly komentované rešerše, současného stavu problematiky, výběru námětu a navrhovaného postupu výuky. V teoretické části práce komentovaná rešerše dělí zdroje do tematických kapitol a shrnuje jejich přínos k problematice. Současný stav problematiky se věnuje v detailu skutečnostem, které ovlivňují návrh postupu výuky.

Vlastní práce obsahuje kapitoly věnující se výběru námětu, kde jsou jednotlivé náměty z teoretické části posuzovány. Na tyto kapitoly navazuje navrhovaný postup výuky, který je zde dělen do lekcí.

2 Komentovaná rešerše informačních zdrojů

Literatura v jednotlivých sekcích je řazena primárně podle roku vydání, od nejstarších po nejnovější publikace. Náměty používané ve výuce uvedené v této literatuře jsou vyvedeny do kapitoly 3.7 Analýza námětů.

2.1 O výuce objektově orientovaného programování

Pro návrh vlastní struktury výuky na základě postupně vylepšovaného programu je důležité věnovat pozornost již existujícím popsaným přístupům a metodám. Protože není dostatek literatury přímo se věnující tématu, jak ho definuje zadání této diplomové práce, je třeba procházet související literaturu. Tato sekce je věnována literatuře na téma objektového programování a metodikám jeho výuky.

Encapsulation and Reuse as Viewed by Java Students [13]

Krátká studie z roku 2001 věnující se vnímání konceptu zapouzdření a znovupoužití kódu studenty. Bohužel zde není zcela jasné, zda studenty vnímaná složitost příkladů byla zapříčiněna omezenými znalostmi a zkušenostmi studentů, nebo zda použité příklady skutečně potřebovaly upravit úrovně růstu jejich komplexity. Tato konkrétní studie se stává již irelevantní, téma, kterému se věnuje, by bylo vhodné opět zpracovat v současných kurzech výuky objektově orientovaného programování.

Teaching Objects-first In Introductory Computer Science [7]

Studie shrnuje slabé a silné stránky použití přístupu Object-First a prostředí Alice ve vstupních kurzech programování na univerzitě. Článek je sice starý (2003) a celkem stručný, ale stále přínosný pro čtenáře, kteří nemají vlastní zkušenosti s výukou programování skrze prostředí typu Alice.

Proč a jak učit OOP žáky základních a středních škol [43]

Článek se zabývá zejména postupem a obsahem výuky žáků základních a středních škol. Klade důraz na objektově orientované programování a snaží se odrazovat od výuky strukturovaného programování, jelikož takový přístup spíše znesnadňuje osvojení objektového programování. Velmi dobře a přehledně strukturovaný a stručný. Vhodný i pro zkušené učitele programování, zároveň jedním ze základních inspiračních vzorů této diplomové práce.

Současné trendy v metodice výuky programování [41]

Tento článek, byť starší, pojednává o posunu v přístupu k výuce programování. Stručně popisuje metodiky používané ve výuce programování a obhajuje přínosy výuky zaměřené na návrhové vzory. Je velmi přínosným úvodem do metodik výuky programování a rozcestníkem pro volbu metodiky.

Teaching Inheritance Concepts with Java [50]

Zabývá se otázkou, zda by se dědičnost měla učit na začátku, nebo by se nejdříve měly použít předpřipravené konstrukce. Vychází z výuky Javy metodou Object-First v prostředí BlueJ na University of Hamburg. Kurz se snaží naučit izolovaně tři témata: rozhraní, polymorfismus a dědičnost. Článek je stručný a návodný pro aplikaci výukového vzoru *Spirála*.

Student Understanding of Object-Oriented Programming as Expressed in Concept Maps [46]

Studie zkoumá znalosti studentů po absolvování vstupního kurzu programování z oblasti objektově orientovaného programování skrze konceptuální mapy. Studenti mají za úkol vyjádřit grafem základní principy a vlastnosti objektově orientovaného programování a graf by měl obsahovat výrazy: objekt, instance a třída. V závěru konstatuje, že vyjádření skrze konceptuální mapy je nejspíš nevhodné, této studii tedy není vhodné věnovat více pozornosti.

Practical Problem-Based Learning in Computing Education [36]

Metoda výuky skrze praktické řešení problémů (PBL) se využívá úspěšně v medicíně. Tento článek navrhuje zařadit přístup řešení problémů do výuky informačních technologií. Odkazuje se na řadu článků o aplikaci PBL do výuky počítačových věd. Může posloužit jako vhodný úvod do této problematiky.

A Model-Driven Approach to Teaching Concurrency [6]

Popisuje přístup k výuce tvorby konkurenčních aplikací, sdílejících hardwarové prostředky. Založeno na šestnáctitýdenním kurzu programování. Studie má jen malý přínos pro výuku programování ve vstupních kurzech.

2.2 O výuce návrhových vzorů

Zvolená metodika výuky, Architecture-first, má za cíl naučit návrhové vzory, neklade důraz na výuku konkrétního programovacího jazyka. Sekce je věnována publikacím popi-

sujícím praxi výuky návrhových vzorů, ideálně v úvodních kurzech programování na vysokých školách.

Teaching Design Patterns in CS1: A Closed Laboratory Sequence based on the Game of Life [57]

Stručný a podrobný článek, popisuje výuku několika vybraných návrhových vzorů na minimalistickém příkladu hry. Dobře využitelné k inspiraci, jak implementovat více návrhových vzorů do jediného příkladu.

Teaching Design Patterns by Stealth [56]

Případová studie, zabývá se výukou návrhových vzorů v navazujících kurzech programování. Aplikuje obecné pravidlo pro výuku návrhových vzorů, je třeba nejprve naučit studenty vzor použít, není ale třeba vzor nutně definovat a pojmenovat. Článek je velmi stručný a věcně argumentuje své stanovisko na konkrétních příkladech.

Teaching Design Patterns Through Computer Game Development [17]

Zde popisovaná výuka je již pokročilým kurzem programování pro všestranně schopné studenty. Vychází z faktu, že dnešní studenti jsou stále více obklopeni hrami, a nejen hry rádi hrají, ale také studují jejich jednotlivé aspekty od návrhu a vývoje, přes jejich grafiku, zvuk či finanční model až po jejich sociální vlivy. Počítačové hry jsou proto ideálním komplexním příkladem s vysokým potenciálem zaujmout většinu studentů. Vysvětluje skupinu návrhových vzorů na komplexním příkladu, ale věnuje se tématu spíše z pohledu komplexního návrhu aplikace.

Teaching Design Patterns Using a Family of Games [18]

Popisuje výuku návrhových vzorů skrze skupinu her se stejným základem, který je neustále rozšiřován a opětovně používán v nových modifikacích hry. Při návrhu praktických cvičení je vhodné tento stručný článek projít, zda nelze některé principy aplikovat i mimo sérii her s podobným základem.

2.3 O výuce počítačových věd

Sekce o výuce počítačových věd obsahuje literaturu, která je stále relevantní pro účel návrhu výuky, jak je popsána cíli této diplomové práce, ale nevěnuje se striktně ani objektově orientovanému programování, ani návrhovým vzorům.

The Psychology of How Novices Learn Computer Programming [32]

Článek je sice z roku 1981, ale zde pospané principy lze aplikovat na výuku libovolné problematiky. Pochopitelně používá příklady, které již dávno nejsou aktuální. Pro účely běžné výuky je tento článek až příliš podrobný.

A constructivist Framework for Integrating the Java Paradigm into the Undergraduate Curriculum [20]

Vydáno v době, kdy Java byla novou technologií a koncept objektově orientovaného programování byl relativně nový. Ukazuje, že znalosti a zkušenosti s procedurálním programováním mají negativní vliv na schopnost studentů učit se objektově orientovaný přístup, a to je jediným přínosem tohoto článku.

On the Quality of Examples in Introductory Java Textbooks [5]

Analýza příkladů v učebnicích ukazuje na obecný stav daných příkladů, ale neuvádí, jak tyto ukázky zlepšovat, nebo čemu se vyvarovat. Tomuto článku vhodné věnovat zvláštní pozornost.

Computer Science Education in Schools [22]

Tento článek z roku 2013 pátral po odborných článcích pro speciální vydání ACM žurnálu (Association for Computing Machinery). Důvodem bylo, že stále více zemí si uvědomuje důležitost vzdělání v oblasti informatiky a mohly by se inspirovat v zemích, které jsou v tomto ohledu napřed. Samotný článek je dnes irelevantní.

A game Engine to Learn Computer Science Languages [48]

Tato studie se skládá ze dvou částí, první popisuje framework, či engine, s jakým se pracuje, druhá porovnává dva způsoby výuky, tedy klasický a výuku skrze hru. Pracuje s příliš malým vzorkem studentům, dosažené závěry nemusí být spolehlivé.

Game-based Learning and Game Construction as an E-learning strategy in Programming Education [37]

Zabývá se výukou na příkladu her, zejména v prostředí e-learningu. Řada motivů lze přenést do klasické výuky na vysokých školách, některé z nich jsou již úspěšně zavedeny. Celkově tento článek nestojí za bližší zkoumání.

Learning basic programming concepts with Game Maker [24]

Popisuje případ zařazení nového celku do výuky programování, konkrétně tvorba hry pomocí Game Makeru. Výuka v tomto podání se nevěnuje objektově orientovanému pří-

stupu a je v tomto ohledu irelevantní. Užitečné mohou být poznatky ze samotného přístupu k výuce a zadání úkolu.

2.4 O podpoře výuky

Zde je uvedena literatura věnující se podpoře výuky programování, kde gamifikace není hlavním a jediným tématem. Zdroje věnující se gamifikaci jsou uvedeny v další sekci.

Computational Tools for Computing Education [54]

Článek stručně kategorizuje nástroje na podporu výuky programování, neuvádí ale příliš konkrétní příklady ani přínosy nástrojů. Užitečný zejména pro málo zkušené pedagogy.

Improving Play and Learning Style Adaptation in a Programming Education Game [31]

Studie řeší problém, že výuka strukturovaného programování pomocí hry často nedosahuje cílových výsledků tím, že se snaží přizpůsobit výuku různým typům studentů. Cílovou skupinou studentů jsou žáci K-12, neboli děti od školky po 12 ročník, v podmínkách České republiky by se jednalo o studenty až do maturitního ročníku. Přesto může být užitečné prozkoumat uvedené typy osobností a jejich přístup k učení se jako další vstup pro modifikaci postupu a formy výuky.

Profile-Based Algorithm for Personalized Gamification in Computer-Supported Collaborative Learning Environments [26]

Řeší otázky návrhu personalizace gamifikačních prvků a jejich vhodný výběr na míru studentům podle jejich herních osobností a přiřazení specifických úkolů jedincům v prostředí kolaborativního učení. Studie, založená na praxi autora, je stručná a přehledná, vyplatí se jí věnovat pozornost.

2.5 Gamifikace ve výuce

Sekce je rozdělena podle zaměření literatury na vážné hry a na gamifikaci. Ačkoli cílem práce není implementovat gamifikační prvky, ani na ně postup výuky připravit, je vhodné brát v potaz praxí ověřené nebo zavržené praktiky, které se nemusí nutně vztahovat na formu cvičení, ale i na přístup k výuce jako takový.

2.5.1 O vážných hrách

Game design for a serious game to help learn programming [45]

Tato práce analyzuje návrhy her používaných pro výuku programování a rozebírá jejich časté problémy, známky dobrého návrhu a požadavky na návrh hry. Je ale spíše přínosná pro návrh kvalitní, atraktivní a robustní hry než pro návrh dobrého příkladu pro výuku.

Jeu sérieux et motivation des étudiants pour apprendre: influence du contexte avec Prog&Play [35]

Zabývá se otázkou motivace studentů a jejich spokojenosti při využití vážné hry při výuce úvodního kurzu programování. Doprovodný experiment proběhl na několika univerzitách. Bohužel je článek příliš stručný a neposkytuje příliš přínosných informací.

Étude de l'intégration d'un jeu sérieux pour l'enseignement de la programmation dans différents contextes universitaires [34]

Rozšiřuje výše uvedenou studii *Jeu sérieux et motivation des étudiants pour apprendre: influence du contexte avec Prog&Play* ([35]), detailněji zpracovává zpětnou vazbu studentů na použití vážné hry ve výuce. Na rozdíl od původní studie, tato je dostatečně detailní a přínosná, problematické pro čtenáře může být, že je dostupná pouze ve francouzštině.

Analysing the Enjoyment of a Serious Game for Programming Learning with Two Unrelated Higher Education Audiences [55]

Zde se analyzuje, jak je přijímán koncept vážné hry pro výuku programování. Případová studie dobře shrnuje příklad i závěry, výstupní data jsou rozebrána detailně a nemusí být relevantní.

2.5.2 O gamifikaci

Climbing Up the Leaderboard: An Empirical Study of Applying Gamification Techniques to a Computer Programming Class [38]

Tato studie vstupního kurzu programování v pythonu se zabývá gamifikací výuky napříč celým kurzem. Pro inspiraci možností využití gamifikace v kurzech programování je určitě velmi přínosná.

Publication trends in gamification: A systematic mapping study [25]

Mapuje, jak se věnují publikace posledních let tématu gamifikace. Největším přínosem této publikace je definice rozlišení vážných her a gamifikace a shrnutí hlavních témat v oblasti gamifikace. Článku tedy není třeba se blíže věnovat.

Scripting Environments of Gamified Learning Management Systems for Programming Education [52]

Existuje řada nástrojů pro podporu výuky programování z hlediska řízení a vyhodnocování, některé více obecné než jiné. Tato studie se snaží identifikovat primární skriptovací prostředí pro systém řízení gamifikované výuky programování, vysvětlit roli skriptovacího prostředí, navrhnout technické řešení a zformulovat klíčové prvky návrhu a jejich dopady. Pro účely této diplomové práce není technické řešení relevantní, pro účely návrhu gamifikované výuky je článek užitečný.

State of the art in Game Based Learning: Dimensions for Evaluating Educational Games [53]

Analyzuje, jaké jsou převážně uváděné přístupy a metody v pracích týkajících se vyhodnocování výukových her. Poznatky této práce jsou specifické pro výuku skrze hry a jsou jen obtížně použitelné v jiném kontextu, není tedy potřeba se jí blíže věnovat.

2.6 O výuce

Do této sekce byly zařazeny zdroje návodné pro výuku obecně, aplikovatelné na libovolné oblasti výuky a problematiky.

What Video Games Have to Teach Us About Learning and Literacy [23]

Velmi detailně rozebírá úspěšné a účinné principy, jakými nás hry učí poznávat komplexní systémy a vstřebávat velké množství informací. Uvádí konkrétní příklady her, ze kterých čerpá. Přestože je kniha z roku 2003, rozhodně se vyplatí se jí věnovat.

Pedagogical patterns: advice for educators [3]

Představuje sbírku pedagogických vzorů s podrobným vysvětlením. Kniha není krátká, představuje systematicky jeden vzor za druhým, ale stojí za to si ji přečíst.

Lecture Design Patterns: Laying the Foundation [27]

Zaměřuje se na sumarizaci základních výukových vzorů, které je dobré primárně aplikovat před přechodem na složitější a komplexnější vzory a praktiky. Řazení a třízení těchto vzorů v daném článku je bohužel velmi nepřehledné a vzory často opakuje, autor jejich rozdělení opět a lépe zpracovává v *Towards a Pattern Language for Lecture Design: An inventory and categorization of existing lecture-relevant patterns* ([28]).

Towards a Pattern Language for Lecture Design: An inventory and categorization of existing lecture-relevant patterns [28]

Navrhuje kategorie dělení výukových vzorů a rozděluje 72 konkrétních vzorů podle specifikovaných kritérií. Nejpřínosnějšími jsou právě nasbírané vzory a jejich dělení podle aplikace.

3 Současný stav řešené problematiky

3.1 Výuka objektově orientovaného programování (OOP)

3.1.1 Postup výuky objektově orientovaného programování

Kapitola vychází z poznatků *Proč a jak učit OOP žáky základních a středních škol* ([43]). Na praxi založený dokument, o výuce programování, definuje řadu zásad pro výuku nezkušených studentů. Snaží se umožnit studentům vytvářet vlastní programy co nejrychleji. Důvod je zde jednoduchý, pokud by student pouze programoval funkcionalitu nebo vnitřní stav fiktivního prostředí, bez jakéhokoli viditelného výstupu, velmi rychle by vyhodnotil kurz jako neproduktivní, nebo problematiku jako příliš složitou a ztratil by zájem pokračovat s kurzem.

Není vhodné používat nevysvětlené prvky jazyka, je zde riziko, že student prvek nepochopí správně, nebo že bude pátrat po jeho dalším využití, aniž by si osvojil základy jazyka, finálně by se mohl pokoušet používat právě tyto okrajově představené prvky nepatřičně. Informace je třeba dávat po malých soustech, programování spočívá v rozkládání problémů na jejich základní prvky, pochopení a vyřešení jednotlivých problémů. Pokud je před studenta předložena příliš složitá úloha, bude potřebovat patřičný čas k jejímu rozložení a vyřešení, přičemž bude velmi složité zajistit, aby danou úlohu řešil správně.

Pokud má studentům poskytnutá informace pomoci řešit určité úlohy efektivně, ale student již zná alternativní, i když méně efektivní řešení, je pravděpodobné, že na první pohled složité řešení vypustí a bude nadále používat postup, se kterým je dobře obeznámen. Studenti by si měli vyzkoušet vytvářet a ladit své programy samostatně, existuje řada přístupů k výuce, student může dostat předprogramovanou strukturu se špatným chováním, aby její chování napravil, může dostat instrukce k vytvoření programu a následně instrukce k opravení, nebo dostane instrukce k tomu, co má vytvořit a poté samostatně ověřit, zda a kde se vyskytly chyby a jak tyto chyby musí opravit.

Jednoznačně nejvyšší přínos má právě situace, kde student samostatně tvoří i opravuje. Může se to zdát velmi náročné, ale přínosy tohoto přístupu převažují jeho náklady. Je vhodné studenta průběžně navádět do situací, kde je nucen znovu využívat již získané poznatky. Pokud se člověk učí aktivitě, potřebuje si postup vícekrát vyzkoušet, než si ho osvojí, a i poté je pro něj velmi důležité svou schopnost občas využívat, aby si jí plně osvojil. Pokud by se student s daným poznatkem setkal pouze jednou, velmi rychle by ho zapomněl, proto stejně jako při výuce matematiky na základních a středních školách, je i v kurzech programování potřeba jednotlivé prvky nastražit dále do programu výuky.

Zabývá se také pozorností studentů. Člověk rád řeší problémy, ale pokud by řešil stejný problém stále dokola, nebo by měl řešit problém, se kterým se nejspíš nikdy nesetká, brzy začne daný příklad znevažovat a ztratí zájem, proto se vyplatí nalézt zajímavý problém. Problém, na který student již narazil, ale zatím nevyřešil, a rád by ho vyřešil, je ve své podstatě výhra, protože zaručuje maximální pozornost a vlastní iniciativu studenta v rámci výuky. Ze stejných důvodů je vhodné omezit balastní informace. S postupně se zvyšující složitostí příkladů se zvyšuje i celkový objem drobných problémů se stejným jádrem, jejichž řešení by pro studenta bylo jednak zbytečnou repetitivní praxí, ale také by ubíralo velké množství z času věnovaného kurzu.

Je žádoucí zadávat netriviální a komplexní úlohy. Výuka by neměla být ani příliš složitá, ani příliš jednoduchá. Student by měl mít stále k dispozici dostatečnou výzvu, v opačném případě přestává věnovat pozornost výuce. Vystavuje se tím riziku vynechání nějaké důležité části výuky, a pak nezbývá než zmeškané dohánět samostatně nebo zdržovat celou skupinu. Je třeba držet se zásad od počátků, přidávání zásad postupně může být matoucí a budit dojem nejasně strukturovaných nebo stále se měnících požadavků, na což studenti reagují vesměs negativně.

Dále navrhuje postup samotné výuky. Základními prvky, ze kterých se vychází, je zahájení výuky hraním si s objekty a až poté jejich vytváření. Aby se předešlo vytváření zbytečné omáčky a výkladu zbytečně složitých prvků na začátku kurzu, se používá již základ aplikace, který je pouze rozšiřován. Pro tyto účely bylo zvoleno vývojové prostředí BlueJ, pro jeho zpracování tříd do diagramů a grafického znázornění vytvořených instancí objektů, z čehož i vyplývá použití programovacího jazyka Java.

Pro výuku se navrhuje nejprve studenty seznámit s vývojovým prostředím, konceptem tříd, objektů, instancí a zpráv. Poté lze představit parametry a návratové hodnoty posílaných zpráv, představit metody a atributy instancí, poprvé navštívit zdrojový kód a definovat vlastní třídu. Návazně se pracuje s konstruktorem a metodami tříd, což je vhodným okamžikem k představení jednotkových testů, ale prozatím bez nutnosti jejich tvorby. Student se v tento moment dostává hlouběji do aplikace a je vhodné navštívit koncept zapouzdření a smysl komentářů a dokumentace.

Následně doporučuje představit lokální proměnné, konstanty, literály, vstupy a výstupy. Představuje metodu toString, inkrementační a dekrementační operátory a tvorbu vlastní testovací třídy, než se pustí do představování několika návrhových vzorů. V této fázi zejména představuje návrhové vzory přepravka, jedináček a tovární metoda s paralelním představením a využitím debuggeru k hledání chyb a ladění programu. Představení rozhraní, jeho dědičnosti a dědičnosti tříd představuje způsob, jakým jednoduše znovupoužít již vytvořený kód a zpřehlednit celou aplikaci před zavedením návrhového vzoru stav a rozdělení celé aplikace do balíčků, následně je možné již velmi jednoduše představit využití knihoven. S blížícím se koncem výuky ještě představuje abstraktní třídy, podmíněné příkazy, výjimky, cykly, kontejnery, pole a závěrem ošetřování výjimek.

Autor zde konstatuje, že na velmi široké škále studentů nejen základních a středních, ale i vysokých škol a profesionálů z praxe, se projevuje vysoká účinnost stejného postupu výuky pro všechny kategorie studentů, a proto nemá valný smysl rozlišovat výukové struktury pro různé posluchače. Samozřejmě je stále potřeba, aby učitel bral ohled na schopnosti a dosavadní znalosti svých studentů, je potřeba podle studentů zvolit náležitou délku výukových celků, tempo výuky i podrobnost výkladu. Dále se ukazuje, že studenti i profesionálové s předchozími zkušenostmi strukturovaného programování mají výrazně větší problémy pochopit a osvojit si objektový přístup k programování než například žáci základních škol bez zkušeností s programováním.

V tomto postupu je kladen důraz na prosazování metodiky architecture-first, která se snaží nejprve naučit správný návrh a návrhové vzory, než se pustí do specifických vlastností daného jazyka. Zejména využití podmíněných příkazů, cyklů a polí se zde řadí na poslední lekce výuky, z jednoduchého důvodu, studenti po setkání se s jednoduchým řešením mají tendenci takové řešení aplikovat na většinu problémů a často pak přechází komplikovanější, ale stabilnější a flexibilnější řešení již vytvořená někým jiným, nebo popsaná návrhovými vzory. Právě tuto logiku následuje tato práce, programátor by s využitím objektového přístupu měl právě rozhodování a cykly rozkládat na jejich drobné prvky a uvést je až v nejnižších sekcích programu, neměl by tvořit rozhodovací pyramidy uprostřed složité třídy a vytvářet tím zbytečné překážky pro návaznou úpravu programu.

3.1.2 Metodiky výuky programování

Kapitola je založena na článku *Současné trendy v metodice výuky programování* ([41]). Sumarizuje používané metodiky. Zdůrazňuje nutnost uvědomění si kontextu problematiky před obsahem samotné výuky. Řada středních škol i dnes stále učí strukturované programování a algoritmizaci, přestože lze tvrdit, že algoritmy jsou hotové a dnes se pouze aplikují na problémy. Proč bychom vymýšleli opět kolo, pokud ho pouze potřebujeme znovu použít. Jak před deseti lety, tak i dnes, se praxe zabývá především implementací, adaptací a zlepšováním stávajících aplikací, případně jejich přenosem mezi různými platformami a technologiemi.

Poukazuje se zde na fakt, že různí pedagogové měli v různých obdobích a podmínkách různé přístupy k výuce. Přístupy jako hardware-first, kde se kladl důraz na technickou stránku elektrotechnických zařízení před jejich programováním, algorithm-first, představující nejprve algoritmizaci problémů před jejím samotným programováním, nebo functional-first, zastánce funkcionálního programování, jsou v dnešní době spíše historickými přístupy. Zastánce si stále drží přístupy object-first, začínající s tvorbou objektů, breadth-first, odkládající programování celkově do pozdějších fází studia, kdy už student má širší poznání problematiky a finálně design-pattern-first, propagující výuku návrhu a návrhových vzorů jako účel učení, na rozdíl od drobností jako jsou jazyková specifika.

Vzhledem k rychlosti vývoje a transformace jednotlivých jazyků a platforem není zcela praktické studenty učit podrobné vychytávky jednoho programovacího jazyka, jednak před jejich nástupem do praxe se může samotný jazyk změnit a přinést lepší principy nebo konkrétní knihovny, ale také se může začít využívat kompletně jiný programovací jazyk. Myšlenka vychází z předpokladu, že ačkoli jednotlivé jazyky se mohou měnit a střídat celkem rychle, samotný objektový přístup se nejspíš za velmi krátké období nezmění, proto má smysl investovat čas do studia právě objektových specifik. A právě příklady z praxe ukazují, že úprava a správa projektu rozpracovaného jiným týmem bývá častou pracovní náplní, zde se také uplatňují spíše návrhové vzory než základní programovací konstrukce.

Samotnou výuku ale nemusíme trávit vysvětlováním návrhového vzoru, je daleko důležitější pochopit co řeší, jaký je jeho přínos a jakým způsobem se aplikuje. Přínosem pro studenta není název daného vzoru, ale jím přidaná hodnota, proto není nutnou podmínkou vzory vysvětlovat. Tato práce si zde bere příklad a snaží se návrhové vzory využívat napříč jednotlivými kapitolami, jejich vysvětlení nechává na pedagogovi.

Použití metodiky vývoje na základě konceptuálních modelů

Studenti v kurzech popsaných v *A Model-Driven Approach to Teaching Concurrency* ([6]) mají za úkol analyzovat zadání, navrhnout aplikaci, vytvořit specifikaci a detailní design a následně aplikaci naprogramovat. Hlavním problémem, který se v kurzu řeší, je sdílení zdrojů a zacházení s nimi.

Identifikuje nevýhody přístupu, zejména obtížnost pro začátečníky, protože vyžaduje znalost modelovacích jazyků, pokročilou znalost programovacího jazyka, základních programovacích konceptů a návrhových vzorů. Tím, že studenti sami aplikaci navrhují a podrobná kontrola jejich návrhů je často nemožná, se stává, že si student práci výrazně ztíží.

Autor řadí mezi nevýhody také automatické generování kódu z připraveného modelu, protože se tím zmenšuje prostor pro vlastní práci studenta. Zde je spíše otázkou, co je cílem výuky, zda procvičovat psaní kódu nebo učit problematiku konkurence, která už tak je spíše pro pokročilé studenty, kteří nemají zapotřebí opakovaně implementovat stejné metody, které mohou být automaticky generovány.

Modelovací jazyky se ve vstupních kurzech obvykle neučí, je ale otázkou, zda by seznámení studenta s konceptem konkurence nemohlo být přínosné již v úvodních kurzech. Například stávající kurzy používající prostředí BlueJ představují koncept modelu programu bez nutnosti znalosti modelovacího jazyka, představený model je dále rozebírán v navazujících kurzech. Podobně by student mohl narazit na příklad konkurenčního přístupu ke zdrojům a tím by se ospravedlnila výuka návrhových vzorů, které právě konkurenci řeší.

Při zkoumání znalostí absolventů vstupních kurzech programování prostřednictvím konceptuálních map ([46]) měli studenti za úkol vyjádřit grafem základní principy a vlastnosti objektově orientovaného programování. Vyplynulo, že studenti mají problémy s pochopením základních konceptů objektově orientovaného programování a v mnoha případech mají i problémy s užitím konceptuálních modelů. Studenti měli za úkol vytvořit graf, který by měl obsahovat výrazy: objekt, instance a třída. Vyjádření je pochopitelně náročnější než pochopení, proto při hodnocení nebyly hledány pouze pojmy, byla snaha rozpoznat koncept v daném diagramu, přestože není jmenován.

Téměř všichni studenti zvládli do konceptuální mapy zařadit koncepty třídy, instancí, metod, atributů a proměnných. Tento fakt neznamená, že konceptům správně rozumí, ale alespoň je spojují s problematikou. Je celkem překvapivé, že objekt, přestože byl jasně zmíněn v zadání, obsahovala jen polovina konceptuálních map. Koncept dědičnosti a zapouzdření nedosáhly většinového použití, přesto byly znatelně zastoupeny. To samé se nedá říct o konceptech polymorfismu, abstrakce nebo modelování, které byly zastoupeny jen vzácně. Zadavatelé úkolu očekávali, že v diagramech najdou koncept posílání zpráv, ten ale nenalezli ani v jednom případě.

Závěrem uvádí ([46]), že studenti mají problémy s porozuměním základním konceptům objekt, instance a třída. Nepodařilo se ve výuce je naučit konceptům zapouzdření, dědičnosti a polymorfismu. Noví studenti kladou příliš velký důraz na detaily a pojmy, nikoli na koncept a možná také proto není použití konceptuálních map za účelem zkoumání znalostí studentů vhodným nástrojem.

Je tedy vhodné se zamyslet, zda konceptuální myšlení nad problematikou je závažným problémem pro studenty vstupních kurzů a jak by se tato situace dala řešit. Je možné, že interakce s konkrétním programovacím jazykem upoutává jejich pozornost na úkor pozornosti ke konceptu a obecnému řešení. V takovém případě by mohlo prospět užití pseudokódu navrženého právě pro demonstraci konceptů.

Použití metodiky object-first v prostředí Alice

V této kapitole se vychází z *Teaching Objects-first In Introductory Computer Science* ([7]). Mezi slabé stránky používání vývojových prostředí typu Alice řadí vynechání výuky syntaxe. Studenti v úvodním kurzu skládají bloky kódu, ale navazující kurz jim představuje reálné vývojové prostředí. Uvádí, že výuka absence přímé výuky syntaxe není zásadním problémem, jelikož se studenti syntax rychle naučí v návazném kurzu bez znatelných obtíží. Druhou slabou stránkou je eliminace chyb programátora. V praxi se setkáváme s chybami plynoucími z implementace, z integrace, z kompatibility nebo jiného softwaru a také ze zadání. Tento fakt by neměl být pro studenty překvapivý, ale měli by se s jeho projevy a následky seznámit raději dříve než později.

Z pohledu autora silné stránky výrazně převažují. Studenti získávají při použití Alice lepší smysl pro návrh. Lépe chápou rozdíly mezi objekty a třídami, jejich kontext je ve vizuál-

ním prostředí výrazně uchopitelnější. Osvojují si lépe metodu pokus omyl, jelikož skládání bloků je oproti psaní kódu časově efektivnější a jednodušší. Program budují v inkrementálních krocích a nesnaží se vytvořit jednu velkou komplexní strukturu, kterou by se následně snažili zprovoznit. Pochopení a použití zapouzdření a dědičnosti je intuitivní. Pozitivní je i rychlé a dobré pochopení chování programu řízeného událostmi.

3.1.3 Výuka dědičnosti a zapouzdření

Na University of Hamburg v kurzech programování ([50]) důrazně aplikují pedagogický vzor *Spirála*, který říká, že je jednodušší se naučit používat nástroj nebo mechaniku než ji tvořit. A že ani samotná tvorba nemusí být komplexní aktivitou a obvykle stačí naučit vybrané aspekty, protože ke zbylým je možné se vrátit až budou potřebné.

Proto nejprve učí koncept, na který navazuje konkrétní implementací. Definuje terminologii, kde koncept je univerzálně platným pro řadu programovacích jazyků, proti tomu mechanika je konkrétní funkcí daného jazyka. Rozlišuje mezi rozhraním jako konceptem neboli množinou metod třídy, a rozhraním jako mechanismem, což je konstrukt jazyka Java. Při obecném popisu používá termín typ pro označení množiny hodnot a operací. Pojem třída je exaktní definicí typu. Operace je abstraktním procesem obecné transformace dat, zpracovávajícím vstupy na výstupy. Metoda je implementací kódu.

Samotná výuka nejprve představuje koncept rozhraní a následně konstrukt v Javě. Navazuje jednotkovými testy a představuje black-box testování, metodu tvorby testů. Pokračuje představením podtypů a dědičnosti. Jako účel dědičnosti představuje redefinování metod. Nabízí příklady úpravy, rozšíření nebo kompletního přepsání metod. Závěrem se věnuje abstrakci typů, vysvětluje, že stejný koncept lze implementovat více způsoby. V abstrakci nezáleží na užití, ale na efektivitě při exekuci, zda se například seznam má řadit až v moment, kdy je potřeba, nebo zda se má řadit při každé změně.

Studie *Encapsulation and Reuse as Viewed by Java Students* ([13]) sumarizuje zpětnou vazbu studentů na výuku problematiky zapouzdření a znovupoužití kódu v jazyce Java. Tato studie byla vydána v roce 2001 a proto je potřeba její poznatky brát s jistou rezervou, jelikož Java i koncept objektově orientovaného programování byly relativně novou problematikou a nestály za nimi roky zkušeností s jejich výukou. I samotné vnímání programování bylo odlišné. Zatímco dnes vidíme programování jako aktivitu, jejímž výsledkem je produkt určený k používání, dříve, jak autor uvádí, bylo programování vnímáno jako aktivita zaměřená na počítač a řešení problému. Tedy chybí zde aspekt orientace na uživatele.

Ukazuje, že schopnost generalizace pomáhá studentům s dobrým užitím zapouzdření a znovupoužití metod, ale často následně narážejí na komplexnost vlastní tvorby. Přecházení mezi více deklaracemi tříd se jeví jako obtížné. Zde obtížnost patrně vychází ze zkušeností

s procedurálním programováním a zažitého konceptu řešení problému, nikoli modelu. Bylo zde také vypozorováno, že zapouzdření výrazně ztěžuje trasování zdroje chyb.

Studentům byly ([13]) předloženy série malých příkladů návrhů programu. V rámci testu měli porovnat modely popisující stejnou problematiku. Tito studenti nebyli schopni poznat dobrý a špatný návrh, ale z jejich zpětné vazby jednoznačně vyplynulo, že nehledě na správnost návrhu, méně tříd a metod je pro absolventy vstupního kurzu přehlednější a snazší na uchopení, přesto ale nezavrhují komplexnější návrhy. Ty se jim obvykle jeví jako sofistikovanější i v případech, kdy návrh obsahuje redundantní třídy a metody.

3.1.4 Výuka skrze řešení problémů

Kapitola vychází z poznatků *Practical Problem-Based Learning in Computing Education* ([36]). Standardní přístup k výuce se snaží předávat znalosti. Přestože je snaha o hlubší zapojení studentů do problematiky, naprostá většina nevyužívá myšlení vyššího řádu, ani pokročilou argumentaci. Výuka na základě řešení problémů propaguje přístup poskytnutí instrukcí, namísto faktů nebo řešení. Ta, jak v autor popisuje, se využívá v medicínské praxi zejména ve vstupních kurzech.

Skupině studentů je představen problém, jejich úkolem je poté přijít s řešením a své řešení prezentovat ostatním. Nepředpokládá se, že by student přišel s ideálním řešením, proto teprve navazuje stěžejní fáze výuky, a to otevřená diskuse. Je třeba brát v potaz, že diskuse mohou být časově náročné. Na druhou stranu jejich přínosem je adaptace studentů na styl učení, na který nebyli zvyklí na středních školách, zlepšuje schopnosti studentů v oblasti myšlení i argumentace a motivuje studenty k lepšímu výkonu.

Popisovaný přístup k výuce je vhodný již pro vstupní kurzy. Kritické myšlení a diskuse jsou spojovány zejména s výukou na vysokých školách. Pokud vstupní kurzy tuto myšlenku u studentů dostatečně nepodpoří, může se stát, a v praxi se stává, že studenti i vyšších ročníků na vysoké škole pouze plní minimální požadavky a nezapojují se do diskusí, nesnaží se samostatně rozvíjet své znalosti a schopnosti. Tedy pokud to časový harmonogram dovolí, měli by být studenti vystavování alespoň moderované diskusi.

3.2 Výuka návrhových vzorů

Návrhových vzorů je velké množství a může být obtížné vysvětlovat rozdíly mezi jednotlivými vzory. Jak popisuje *Teaching Design Patterns By Stealth* ([56]), hlavní je princip a přínos řešení, nikoli detail typu, zda se jedná o vzor *Adaptér* nebo *Fasáda*, či rozlišení případů vzorů *Tovární metoda* a *Abstraktní tovární metoda*. Vzory učí na příkladu celkem jednoduché praktické aplikace pro výpočet mzdy, neustále zadává změnové požadavky. Přínos kurzu shrnuje do čtyř rad. Aplikace se mění, je třeba její změny očekávat a připravit

se na ně. Navrhují se aplikace způsobem umožňujícím jednoduché nalezení místa změny. Zásahy do aplikace by se měly týkat pouze jednoho místa, neměly by být rozptýleny po celé aplikaci. Chyby, zavlečené do programu, se mají projevovat pouze ve změněné části programu.

Poslední zmíněná rada nezní příliš jako rada, samozřejmě by to mělo být cílem programátora, ale není jasné, jakým způsobem se zavlečení chyb do jiných částí programu předchází. V praxi zajištění kvality a testování softwaru se provádí regresní testování pro odhalení regrese, tedy zavlečení chyb do již funkčních oblastí programu, regrese ale nevznikají pouze ve změněné části programu, někdy se projeví i v oblastech, které by neměly být změnou nijak ovlivněné. Samozřejmě příčinou regrese v programu je obvykle právě špatný návrh a je dobré vést studenty k dobrému návrhu, ale prozatím neumíme dobrý návrh ověřit bez testování. Proto by jisté regresní testování v rámci výuky programování mohlo být přínosem upravující tvrzení o výskytu chyb na realistickou úroveň.

Studie z University of Wisconsin-Eau Claire ([57]) představuje výuku vybraných návrhových vzorů ve vstupních kurzech programování na příkladu počítačové hry. Zde jsou identifikované tři obecné zásady výuky programování, spolu se čtyřmi specifickými zásadami pro výuku návrhových vzorů. Jednak je vhodné použití grafického výstupu aplikace, studenti nemusí nutně grafickou část aplikace sami implementovat, pouze stačí ji ovlivňovat vlastním kódem a sledovat dopady svého jednání. Doporučuje se využívat experimentace, náhody a překvapení.

Tvorba hry je mnohdy jako malování obrázku, každý má jiné zkušenosti, vize a cíle, pokud jsou studenti omezováni ve své tvorbě, mohou ve vedené úloze ztratit zájem a soustředit se na vlastní neomezované zájmy. Také hry, jejichž výsledek je silně variabilní, často nemají jednoznačné řešení a lze jejich průběh postupným vývojem stále upravovat, aniž by hra rychle ztrácela své kouzlo. Úkoly předložené studentům by měly být inspirovány reálným světem, nemělo by se jednat o abstraktní úlohy. Konkrétní úlohy se dají daleko lépe zpracovat a pochopit, zatímco abstraktní úlohy běžně příliš neinspirují. Zde je dobré podotknout, že hra užitá studií ([57]) právě tento bod nesplňuje a řadí se spíše k abstraktním úlohám.

Pro výuku návrhových vzorů se snaží ([57]) využívat klasické dobře známé úlohy jako základ. Nepředstavuje naprosto nový požadavek vytvořený na míru danému vzoru. Toto je obecně platné pravidlo pro výuku. Poskytnutí řešení problému, o které nikdo nestál v první řadě, nebude studenty adoptováno a bude potřeba jej opětovně představit ve vhodnou chvíli.

Návrhové vzory se také doporučuje zjednodušit na nutné minimum neboli ukázat, co dělá daný návrhový vzor vzorem. Popsaných návrhových vzorů nejsou jenom desítky a bohužel mezi některými návrhovými vzory se jen obtížně hledají rozdíly, aniž by byly představeny komplexnější příklady a případy užití. Pokud se má student nějaký návrhový vzor naučit, měl by se tedy naučit jeho podstatu. Je nutné návrhové vzory vybírat. Vzít sbírku nejdůle-

žitějších čtyřiceti návrhových vzorů a předložit ji kompletně studentům jako nástroj není příliš praktické, je třeba jejich přínos znázornit.

K demonstraci návrhových vzorů je lepší dojít než ji pouze odprezentovat. To znamená, že by problém mohl být již vyřešený nebo v řešení, v moment, kdy se navrhne řešení na základě daného vzoru, student, aby dříve vytvořenou chybu nebo rigiditu odstranil, je nucen použít redaktorování a kompletně tak projít dobrý i špatný návrh a přechod mezi nimi.

Hra života ([57]) obsahuje mřížku bílých a černých polí (buněk), které se s každým krokem aplikace mohou měnit (vznikat a zanikat). Pravidla vzniku a zániku se mohou upravovat. Na této velmi jednoduché hře je na konci semestru představeno a implementováno pět vybraných návrhových vzorů.

Nejprve používá vzoru *Pozorovatel* k oddělení uživatelského rozhraní od modelu hry, studenti implementují metody připojení a odpojení uživatelského rozhraní, notifikace o změně a metodu aktualizace. Následně se používá vzor *Stav* k odlišení funkcionality živých a mrtvých polí, která se během běhu aplikace stále mění. Studenti zde implementují funkce vzniku a zániku, efektivně se tím aplikace zbavuje potřeby podmíněného příkazu v této metodě a usnadňuje rozšiřování možných stavů. Bohužel, nezdá se, že by tento přínos byl patřičně demonstrován.

Na vzor *Stav* navazuje vzorem *Jedináček*, veškeré buňky jsou chováním identické, aby se zbytečně negenerovaly stále nové instance jednotlivých stavů, upravují se zde stavy podle vzoru *Jedináček*, implementací metody pro vytvoření a získání instance stavu. Ačkoli to studie ([57]) nezmiňuje, používá uvnitř vzor *Jedináček* i vzor *Líná inicializace*, tedy vytvoření dané instance teprve v moment její potřeby. Taková funkcionalita přirozeně využívá podmíněný příkaz, který se tato diplomová práce snaží odsunout do pozdějších týdnů výuky, a kterému je možné se v tento okamžik za jistou cenu vyhnout.

Dalším použitým vzorem je *Příkaz*. Jelikož postupné vyhodnocování jednotlivých buněk by ovlivňovalo výsledek kroku, využívá se *Příkaz* k vytvoření jednotlivých rozhodnutí s odloženým provedením. Studenti vytváří konstruktory příkazů a metodu jejich provedení. Návazně se využívá vzoru *Návštěvník*, aby se delegovalo rozhodnutí o použití dříve vytvořených příkazů, není potřeba aplikovat příkaz znič na již zničený předmět, studenti proto vytváří metodu pro akceptování příkazu a jeho delegování až do relevantní třídy.

Využití demonstrovaného postupu v této diplomové práci je limitováno odlišným přístupem, zatímco studenti z University of Wisconsin-Eau Claire se nejprve učí jednotlivé programovací struktury, zde je cílem od těchto struktur, pokud možno, abstrahovat. Návrhové vzory jsou brány jako prioritní obsah výuky. Proto je zde nejpřínosnější užití vzoru Stav k prevenci tvorby pyramid podmíněných příkazů. Přístup architecture-first jednoznačně prosazuje využití návrhových vzorů k prevenci špatného návrhu, s čímž je spojeno riziko zpoždění uvědomění si přínosu návrhového vzoru ze strany studenta.

Jedním ze vstupních předpokladů *Teaching Design Patterns Through Computer Game Development* ([17]) je, že studenti nemusí dobře pochopit návrhový vzor, pokud je demonstrován na izolovaném případě. Přesto se této praktiky nezbavuje a před aplikací vzoru ve výsledné hře je vzor vyzkoušen na malém izolovaném projektu.

Dalším vstupním předpokladem je, že student, aby si řešení problému osvojil, musí nejprve problému čelit a jeho řešení hledat. Pokud je nástroj řešení představen studentovi předem, jeho aplikace se zdá triviálním úkolem, zejména proto, že nebylo třeba vyvinout žádné úsilí k jeho nalezení. Vnímaná důležitost a přínos daného nástroje je pak výrazně nižší.

Kurz ([17]) bere ohled na jednotlivé fáze učení návrhových vzorů, jako nástrojů či metod řešení problémů. Student musí vzoru nejprve porozumět, aby byl schopen vzor kopírovat do své práce. Pokud se naučil vzor začlenit do svého projektu, potřebuje se ještě naučit vzor adaptovat na odlišné podmínky. Až teprve po zvládnutí pochopení, kopírování a úpravy vzoru, si ho student může patřičně osvojit. Celý tento proces trvá běžně tři týdny výuky, proto se doporučuje nepředstavovat zásadní koncepty v posledních třech týdnech před závěrečnými testy. Co studie neuvažuje je, že by se tyto tři týdny daly vyjmout z výuky, skrz samostatnou hodnocenou práci studenta, nutnou pro připuštění k závěrečným testům, na jejímž příkladu by si student mohl poslední látku osvojit.

Neklade se zde ([17]) za cíl, aby se studenti naučili vytvářet komerční hru, přece jen komerční hry mají jiné cíle jako je maximální efektivita programu a vysoké využití výkonu počítače skrz specifika a triky jednotlivých jazyků či platforem. Kvalitní hry se soustředí na zobrazovací schopnosti aplikace, kdy se snaží umět generovat více snímků, než zvládne daný monitor zobrazovat, využívají aktivního i pasivního renderování, bufferování a zahazování snímků pro lepší kvalitu obrazu. Postupem času stále více z těchto funkcionalit přechází na engine dané hry, který si zřídkakdy vyvíjí podnik samostatně, obvykle používají komerčně dostupné nástroje. Proto v očích pedagogů tyto cíle herních vývojářů nepůsobí přínosně pro výuku, jelikož se student v moment příchodu do praxe nejspíš setká se zásadně odlišnými podmínkami. Namísto toho se soustředí na návrhové vzory pro zvýšení přehlednosti programu a jeho upravitelnosti. Sice jdou návrhové vzory často proti cílům dobrých her, tedy příliš nepomáhají výkonu nebo ho přímo snižují, ale cílem je naučit dobré zásady, nikoli naučit tvořit kvalitní hry.

Je patrné, že pozornost studentů při výuce je jedním z problematických aspektů výuky v popisovaném kurzu. Jako jeden z problémů existujících her na jejichž základě se programování učí, identifikuje autor jejich značnou omezenost po stránce výkonnostní i grafické. Tento problém řeší převedením hry na více vláknovou aplikaci tak, aby grafická a logická část aplikace byly řízeny separátně. Rozdělení vláken je dozajista přínosné, může ale výrazně zvýšit buď vstupní náročnost kurzu, nebo náročnost přípravy materiálů k výuce.

Hra Every Extend ([17]) je založena na jednoduchém motivu, hráč ovládá figurku, vyhýbá se pohyblivým překážkám a sbírá bonusy. Při dotyku s překážkou hráč exploduje, ztrácí

život a ničí se veškeré přítomné překážky. Podle zničených překážek získává skóre, dostatečné skóre také přiděluje životy zpět, hra se graduálně ztěžuje.

Studie ([17]) není příliš konkrétní ohledně aplikace návrhových vzorů, ani časového plánu jejich výuky, pouze určuje, které vzory využívá a k jakému efektu. Vzor *Stav* zde upravuje chování překážek zničených a celých, nenavazuje však vzorem *Jedináček*, namísto toho definuje stavy jako vnitřní třídy k omezení tvorby jejich instancí. Vzor *Fasáda* je implementován k umožnění změny grafického rozhraní aplikace za běhu programu. Za použití vzoru *Pozorovatel* se přesouvají některé výpočetní operace na jiné vlákno. Vzorem *Strategie* se deleguje animace na patřičný objekt. Vzor *Návštěvník* upravuje chování při kolizi různých objektů. Závěrem je použit vzor *Jedináček* na přehrávač zvuků.

Jisté aspekty této studie jsou dobrou inspirací zejména zdůraznění, že je třeba brát ohled na samotné učení studentů, nelze předpokládat, že na první pokus vše pochopí a zapamatují si, jak se vzory pracovat. Také, že je někdy vhodné se vyhnout běžným učebnicovým hrám, které jsou technicky omezené a hledat vlastní kompromis mezi dobrým výukovým příkladem a atraktivní hrou.

3.2.1 Výuka návrhových vzorů na skupině her se stejným základem

V této kapitole se vychází z *Teaching Design Patterns Using a Family of Games* ([18]). Na příkladu pokročilého kurzu programování z University of Madrid studie ukazuje možnosti výuky návrhových vzorů za pomoci skupiny her se stejným základem. Pro výuku se obvykle vyberou různé variace her se stejným základem, ať už se jedná o piškvorky, nebo šachy, pokaždé se postupuje od jednoduchých verzí ke složitějším.

Výuka se řídí několika předpoklady, jedním z nich je, že studenti potřebují poznat špatný návrh, aby docenili dobrý návrh. Dalším předpokladem je, že studenti potřebují malé úlohy, aby cítili potřebu návrhové vzory prakticky použít, s čímž nelze jednoznačně souhlasit. Z praktického hlediska, pokud se zjistí, že návrh řešení dané malé úlohy není ideální, nebo je přímo špatný, často se pod časovým nátlakem pouze vytvoří provizorní řešení pro konkrétní úlohu a příště se na základě chyb návrh opraví a vytvoří kompletně nový. Při popisované výuce si tento problém zjevně uvědomili, a proto tlačí studenty do řešení těchto jednoduchých úloh v rámci jediného projektu. Studenti musí vytvořený kód opakovaně využívat a jeho vady si neustále připomínat a řešit.

Kurz omezuje studenty krátkými časovými limity, aby se museli rozhodnout, zda vytvoří funkční řešení včas, nebo dobré řešení později, přičemž příliš pozdní řešení je silně penalizováno. Použitá argumentace je zde taková, že tvorba z průběhu semestru je předmětem závěrečné zkoušky a studenti, kteří tráví více času přípravou, by měli snazší zkoušku.

Ačkoli takový kurz má bezpochyby jisté rysy z praxe a je v tomto směru pro studenty přínosný, jeho pojetí spíše potlačuje osvojení dobrého návrhu. Student si po absolvování daného kurzu zřejmě dobře uvědomuje, proč jsou dané návrhové vzory důležité a proč je

důležité mít dobře navrženou aplikaci, ale na druhou stranu si uvědomuje, že potřebný čas není vždy luxusem, který si může dovolit. Výsledkem může být, že se daný student v praxi rozhodne tvořit namísto návrhu, a jak to, tak s projekty bývá, následné změny ho budou nutit vytvořený kód stále narychlo ohýbat.

Ideální přístup by byl začít analýzou projektu, následuje navržení aplikace a poté její zhotovení, úprava, testování a vypuštění. Lze si ale představit, že stejný projekt se bude někdo snažit řešit agilním přístupem, proto si zjistí něco málo o projektu, vytvoří své řešení a následně komunikuje se zákazníkem o nutných změnách, tento postup se opakuje, dokud jedna strana nevzdá své ambice a projekt je často v pochybném stavu dokončen.

Ve studii popisovaný kurz je určen značně pokročilým studentům, proto požaduje vytvářet hru dostatečně jednoduchou a bez vlivu náhody, aby bylo možné řešit i strategii hry a umělou inteligenci zastávající protihráče. Výuka vede studenty k implementaci návrhových vzorů *Pozorovatel* a *Model-Pohled-Ovládání* v jedné z her, vzory *Strategie*, *Šablonová metoda* a *Tovární metoda* v druhé. Následně ověřuje kvalitu návrhu třetí hrou, respektive návrh patřičně upravuje. Na konci studenty staví před problém dodatečné implementace umělé inteligence a síťové komunikace.

3.3 Přístupy k výuce

Jak ukazuje *A constructivist Framework for Integrating the Java Paradigm into the Undergraduate Curriculum* ([20]), znalosti a zkušenosti s procedurálním programováním mají negativní vliv na schopnost studentů učit se objektově orientovaný přístup a značně znesnadňují aplikaci smysluplného učení.

Proto definuje kroky, které je třeba provést, a praktiky k zavedení, aby se posílil vliv objektově orientovaného programování. Znalosti objektového přístupu je potřeba neustále upevňovat, tedy nestačí jeden kurz programování, ale koncept by se měl projevovat napříč více kurzy. Aby si studenti nový přístup zažili, je třeba více podpořit praktická cvičení a řešení realistických zajímavých problémů.

Prvním krokem by měla být identifikace znalostí studentů. Pokud je to možné, tak rozdělit zkušené a nezkušené studenty, protože méně zkušení studenti se snáze naučí nový koncept. Případně je nutné adaptovat výuku znalostem studentům a podpořit dobré zásady, mezi které se zde řadí preference správného konceptu před faktickém řešením problému.

3.3.1 Smysluplné učení

Kapitola shrnuje poznatky *The Psychology of How Novices Learn Computer Programming* ([32]). Smysluplné učení definuje jako proces propojení nových informací s existujícími

znalostmi. Pro tuto formu učení je zcela zásadní, aby student v první řadě nové informace přijal do krátkodobé paměti. Následně si musí vybavit nějakou související tématiku, nejčastěji tím bývá příklad z praxe. Souvislost nových informací se známým kontextem si musí ve vlastní krátkodobé paměti zformulovat a vstřebat do dlouhodobé paměti. Znalosti vytržené z kontextu, které se dostanou do dlouhodobé paměti, jsou obvykle zapomenuty nejdříve. Typickým příkladem zavádění bezkontextních informací do dlouhodobé paměti, je učení se správných odpovědí nazpaměť. Student při takovém učení vynechává z důvodu náročnosti složku kontextualizace a jeho snaha je neefektivní a nesmyslná.

Dobrým příkladem podpory smysluplného učení je výuka matematiky na základních a středních školách. K její úspěšnosti přispívají zejména dva prvky. Prvním je používání reálných příkladů, slovních úloh pro pochopení postupu a účelu, oproti poskytování vzorců, kterými studenti mohou bez větší námahy dojít ke správné odpovědi.

Druhým prvkem, který řada učitelů matematiky úspěšně aplikuje, je zahájení výuky tematické oblasti úvodním souhrnem kontextu. Často se ve výuce matematiky přechází spirálovým výukovým vzorem od jednodušších elementárních konceptů a úloh, k jejich rozvinutějším a specializovanějším verzím. Pro tuto výuku je typické, že se k dané tématice stále vrací. Pokaždé je nutné studenty uvést do problematiky a ověřit si, že stále znají, co se již učili, a představit jim návazné koncepty a metody, které se teprve učit budou.

Další aspekt výuky, kterému je třeba věnovat pozornost, je metoda poznávání. Zde se definují dva přístupy, black-box a white-box. Black-box představuje situaci poznávání, kdy chápeme problematiku jako uzavřený celek, bez znalosti jejího vnitřního fungování, pravidel a zásad. Tento přístup je často popisován jako badatelský, bohužel ve výuce mají studenti tendenci se touto metodou učit správné odpovědi, nikoli látce porozumět.

White-box, nebo také glass-box, je přístup, ve kterém má student k dispozici náhled do vnitřního fungování, pravidel a zásad zkoumané problematiky. Jak autor uvádí, a co je v dnešní době stále více platné, úroveň vnitřního detailu, je pro účel výuky až příliš rozsáhlá. Je potřeba tento detail redukovat, aby student nebyl zahlcen. Dnes již běžně používáme termín gray-box k popisu logických celků, které zkoumáme s účelným omezením rozlišení jejich vnitřní struktury.

Na závěr uvádí praktiky, které navzdory všeobecně akceptované domněnce nejsou pro výuku prokazatelně přínosné. Jednou takovou praktikou je kladení důrazu na gramatickou přesnost a formulaci výroků. Často bývali studenti různě motivováni si zapamatovat výroky doslovně, jelikož doslovně zopakovaný výrok byl správný. Ale schopnosti zopakovat výrok, schopnosti mu porozumět nebo identifikovat nuance při upravení daného výroku jsou na sobě navzájem nezávislé a mají tendenci se spíše navzájem vylučovat než naopak. Na tuto skutečnost se řada pedagogů snaží reagovat tím, že nechává studenty látku formulovat vlastními slovy. Což jako disciplína je sice pro studenta náročnější, ale stává se, že je student s vlastní zjednodušenou a nepřesnou formulací natolik spokojen, že si jí zapamatuje jako korektní a je potřeba velkého úsilí tento omyl napravit. Ani tento přístup nemá

prokazatelný přínos pro výuku. Formulace vlastními slovy dozajista není vadou výuky, ale neměla by být po studentech explicitně požadována.

3.3.2 Skládání forem výuky

V případě, kdy se přebírá nebo inspiruje výuka z jiné školy nebo země, je potřeba ohlídat si řadu faktorů, které mohly ovlivnit formu a směr přebírané výuky. Tyto faktory lze vyčíst z *Computer Science Education in Schools* ([22]). Zásadním faktorem je systém školství a jeho jak celková úroveň, tak úroveň konkrétního studijního cyklu. Metody používané na vysokých školách nemusí být praktické aplikovat na středních nebo základních školách. Sociokulturní aspekty také mohou ovlivnit úspěšnost metody výuky – faktory jako historie ICT (Information and Communication Technologies), povědomí a obecná znalost technologií, rozdělení věkové, etnické, úroveň socializace a další. Výzkum, financování, řízení kvality výuky a kvalifikace učitelů může implementaci některých metod značně omezovat. Při analýze používaných metodik je třeba znát jejich kontext, tedy motivaci studentů, studijní plán a cíle výuky, zkušenosti studentů a metody výuky typické pro danou skupinu studentů.

A game Engine to Learn Computer Science Languages ([48]) porovnávaná výuku tradiční, která spočívá v přednáškách a v poskytování připravených prezentací, bez praktické složky programování, experimentální výukou, kde staví studenty před hru Lost in Space. Úkolem studentů je pomocí pseudokódu navigovat postavičku k vyřešení daného logického úkolu, tedy musí projít z bodu A do bodu B.

Experimentální výuka v tomto případě ([48]) neposkytuje žádné informace studentům, pouze jim poskytne hru a nechává je její prostředí samostatně prozkoumat a pochopit. Hypotézou zde bylo, že efektivita výuky není stejná při použití tradičního přístupu jako při použití experimentálního přístupu. Teoretickými testy byly měřeny vstupní a výsledné znalosti studentů obou kurzů, nabyté znalosti byly v obou případech statisticky jen nevýznamně odlišné, proto byla vstupní hypotéza vyvrácena, efektivita popsané tradiční a experimentální výuky byla stejná. Vedlejšími poznatky studie bylo, že studenti experimentálního kurzu byli výrazně motivovanější, ale že experimentální výuka nedokáže naučit syntaxi jazyka.

Je nutné konstatovat, že podle zvoleného způsobu tradiční a experimentální výuky nejsou výsledky studie ([48]) překvapující a spíše dokazují potřebu kombinace přístupů, jinak jsou popsané přístupy samostatně srovnatelně neefektivní. Programování je praktickou činností, pouze teoretická výuka bude mít minimální efekt, podobně praktické cvičení bez objasnění konceptu programování je velmi problematické na pochopení a výstupy takového učení nemusí dosahovat očekávané kvality. Aby se dalo na výsledky této studie spoléhat, musela by testovat větší vzorek možností a studentů, zejména kombinaci tradiční výuky s praktickými cvičeními a experimentální výuky s teoretickými základy.

Současná digitální generace studentů má jiné vnímání technologií a také jiná očekávání. Různé MOOC (Massive online open courses) a jiné typy online výuky proto mívají problémy s udržením studentů v kurzech. V *Game-based Learning and Game Construction as an E-learning strategy in Programming Education* ([37]) autor uvádí, že k závěrečnému testu dojde pouze polovina studentů. Proto je snaha zvýšit zájem studentů skrze počítačové hry.

Studie ([37]) identifikuje čtyři způsoby, jakými se učíme z her. Naučné hry na míru se již běžně používají na řadě základních škol a prokazují značné prodloužení doby, po kterou je dětská pozornost udržitelná. Komerční hry mají také potenciál své hráče něco naučit, zejména se to týká strategického rozhodování, porovnávání přínosů a ztrát, řízení rizik nebo i jenom jemné motoriky potřebné k ovládání dané hry. Hry na podporu interakce mají za cíl podporovat sociální dovednosti jednotlivých účastníků, znatelně pomáhajíc v tomto ohledu slabším jedincům se naučit interakci s okolím, v promíjejícím anonymním online prostředí. Ale zásadní způsob výuky skrze hru pro kurzy programování je právě samotný vývoj hry, kde se nemusí nutně jednat o tvorbu moderního titulu za použití nejmodernějších technologií ani originálního nápadu.

Napříč více běhy zkoumaného kurzu ([37]) bylo vyzkoušena a statisticky vyhodnocena výuka programování na příkladu praktické aplikace a na příkladu hry, zásadním rozdílem zde byla účast studentů v kurzu. Ukázalo se, že část studentů se snaží kurz pouze dokončit, nevyužívají tedy potenciálu zadaných úkolů a plní pouze dané minimum. Striktnost zadání a požadavků na výsledek má u takových studentů šanci zvýšit přidanou hodnotu kurzu, ale může jít proti jejich vůli.

Na druhou stranu striktnost zadání potlačuje kreativitu a ta byla zvolena jako jeden z hlavních cílů kurzu. Hledá se ([37]) proto vhodná rovnováha mezi striktností a volností zadání, aby zároveň kurz neodrazoval studenty, ale zároveň jim dovoloval a motivoval je k tvorbě kvalitních a originálních aplikací. Ukázalo se také, že v omezeném čase se kvalita s originalitou navzájem omezují, proto se přišlo s jasnějším a striktnějším zadáním, navrhující pouze několik námětů her, pro omezení originality, čímž stoupla kvalita finálních aplikací.

Samotná výuka ([37]) má několik charakteristických rysů. Aktivita studentů je řízena, aby se drželi předloženého úkolu a neodtrhli se od směru výuky. Studentům je poskytnut dostatečný čas na úvahu nad jejich projektem a průběžně je jim poskytována zpětná vazba k jejich práci. Studenti musí mít možnost řízení vlastních projektů, není žádoucí jim vše diktovat, ztráceli by zájem o kurz. Nechává studenty připravit se na další úlohu předem, umožní rozmyšlení a přípravu dané úlohy, student poté podává lepší výsledky. V těchto kurzech není cílem studenty stavět před problém a sledovat jejich okamžitá řešení.

S postupem času se začíná o online kurzech mluvit jako o potenciální konkurenci vysokým školám, některé univerzity již nabízejí z velké části studium online. Ačkoli se zatím zdá celkem nepravděpodobné, že by vysoké školy mohly ztratit svou váhu, přece jen je vhodné sledovat trendy i v takové výuce a hledat dobré přístupy, které lze aplikovat i ve formě

prezenčního studia. Atraktivita tématu pro studenty je jen jednou z komponent výuky, ta již byla vysokoškolskými kurzy dobře přijata. Za zamyšlení stojí řízení kvality a originality práce skrze striktnost zadání. Zde v praxi bývá největší problém v rozdílném přístupu studentů, jak na to již tato studie narážela, zaujatí studenti nemají problém nalézt a rozvést vlastní námět. Problém je ta část studentů, kteří plní pouze minimální požadavky, nebo je pouze nezajímá navržené téma. Takoví studenti často potřebují přesné zadání, kterého se mají držet.

3.3.3 Příklady v učebních materiálech

Kvalita ukázek, čerpaná z učebnic a příruček pro výuku programování v jazyce Java je hodnocena v *On the Quality of Examples in Introductory Java Textbooks* ([5]). Kvalitu ukázky staví nad kvalitu jiných aspektů výuky, z důvodu, že studenti obvykle psané poučky i výklad ignorují a zapomínají, ale k ukázkám se vrací a nejčastěji je pouze reprodukují. Proto zlepšení příkladu bude mít největší dopad na zlepšení kvality výuky a mělo by mít vysokou prioritu. Z analýzy příkladů autor vyjímá zjednodušené příklady, postupně vylepšované programy, online tutoriály a publikace vydané před rokem 2007. Označuje je za nereprezentativní. Ačkoli se tato diplomová práce zaměřuje právě na výuku skrze postupně vylepšované programy, je stále možné využít tuto studii k identifikaci klíčových aspektů ukázek a jejich častých nedostatků.

Podle autora ([5]) by učebnice programování v Javě měla pokrývat následující témata: analýza problému, testování a debugging, aplikace konceptu v praxi, relevantní příklady včetně kontextu, algoritmizace, syntaxe a grafika. Dále definuje nároky na jednotlivé ukázky, které ve výzkumu bodově hodnotí. Jedná se o bezchybnost příkladu s kompletním příkladem. Návrh by měl být dobře čitelný a formátovaný. Má používat rozumné abstrakce, chování a vztahy tříd. Kód by měl být objektově orientovaný a měl by propagovat objektově orientované myšlení. Ukázky musí být účelné s vyváženou informační náloží. Finálně by zde měl být popsán zejména proces dosažení řešení, nikoli pouze výsledný stav.

Bodové hodnocení autor ([5]) uvádí průměrné za všechny analyzované ukázky a neuvádí, jak by bylo vhodné nedostatky konkrétně řešit. Nejlépe hodnocené aspekty jsou bezchybnost, kompletnost, čitelnost, formátovanost a použití rozumných vztahů tříd. Aspekty hodnocené neutrálně, které nemají zásadní problémy, ale daly by se vylepšovat, jsou rozumnost abstrakce, ukázkovost použití objektové orientace, účelnost ukázky a vyváženost informační nálože. Zbývající aspekty shledává autor jako zanedbané, nejhůře je na tom propagace objektově orientovaného myšlení.

Rady pro návrh frameworku, se kterým studenti v kurzech programování pracují, definuje *A game Engine to Learn Computer Science Languages* ([48]). Uvádí požadavky dvojího typu, požadavky na model hry a na architekturu enginu. Model hry musí studentům umožnit zasahovat do kódu hry. Hra má být závislá na rozhodování, nikoli na reakcích, není zde

potřeba pro hry běžící v reálném čase, tahové hry úplně stačí. Měly by být odděleny uživatelské vstupy od mechaniky hry.

Hra ([48]) by měla být rozdělena do úrovní, jednotlivé úrovně se postupně ztěžují. Cíl hry má být jasně daný, student by neměl mít možnost ztrácet čas zkoumáním hry. Systém výpočtu skóre za splnění úrovně pro zvýšení soutěživosti studentů. Samotný engine by měl zpracovávat jádro hry, tedy fyziku a grafiku. Měl by poskytnout studentům množinu základních funkcí. Pokud se vyučuje na základě pseudojazyka, je potřeba implementace interpretéru. Engine také musí podporovat tvorbu zmíněných úrovní.

3.3.4 Rozsah příkladu

Programovací struktury a návrhové vzory jsou univerzální abstraktní struktury, pro jejich výuku lze používat velké množství implementací. Konkrétní implementace, příklad, na kterém se problematika ukazuje, může být maximálně zjednodušený nebo relativně komplexní.

Velmi jednoduché na přípravu je použití minimalistického příkladu pro demonstraci problému a jeho řešení. Výuka se pak skládá z řady navzájem nepropojených příkladů. Návaznost a kombinace jednotlivých řešení nemusí být jasná a může být třeba zařadit extra příklady pro hlubší prozkoumání problematiky.

Alternativou je použití jedné tematické domény při přípravě praktických příkladů. Propojení jednotlivých sekcí vychází z podstaty příkladu, případně lze konkrétní příklady rozšířit a vysvětlit jedním příkladem více problémů. Finálně je také možné řešit stále jeden problém do větší hloubky komplexity úpravami původního řešení.

Vzájemné vazby a možnosti kombinace jednotlivých konceptů je také možné zakomponovat do jednoho příkladu, který se postupně vylepšuje v průběhu kurzu. Použití jediného příkladu je náročnější na plánování lekcí a jejich organizaci. Výhodou je, že je problematika celkově propojenější a usnadňuje tak efektivní učení.

3.3.5 Úroveň propojení prací studentů

Je vhodné koncipovat praktická cvičení podle cílů kurzu. V některých případech je vhodná více spolupráce, jinde soutěživost. Zde jsou uvedeny základní typy prací podle jejich návaznosti na práce ostatních studentů.

- **Samostatné práce** – každý student odevzdává vlastní práci, spolupráce nelze připustit. Obvykle zakončeny obhajobou práce. Vhodné pro podporu výuky základů dané problematiky.

- **Volně řešitelné práce** – studentům se umožňuje pracovat samostatně či ve skupinách. Týká se zejména prací, kde je důležité dosažení cíle, nikoli zvolená metoda. Například integrační práce nebo nasazení a konfigurace aplikace na serveru.
- **Skupinové práce** – vyžadují, aby studenti pracovali ve skupinách. Skupina se musí shodnout na jednotném výstupu práce, rozšiřuje komplexitu tématu o komunikační úroveň. Dovoluje studentům se více věnovat problematice jako celku a jejím hlavním otázkám, bez potřeby detailního zkoumání základů, na kterých jednotlivé otázky leží. Zpracování těchto základů si obvykle v týmu studenti rozdělí a seznámí se s výstupy.
- **Závislé konkurenční práce** – výsledné práce jsou v rámci kurzu porovnávány přímou interakcí, tedy v případě skriptu, běží paralelně. Hodnocení prací se odvíjí od jejich úspěšnosti vzhledem ke konkurenci z daného kurzu. Příkladem jsou skriptované souboje ve hře, nebo návrhy projektů. Cílem není pouze řešení úlohy, ale dosažení kvalitního řešení.
- **Nezávislé konkurenční práce** – používá se obvykle pokud výstupy studentů nemohou spolupracovat a je třeba je hodnotit jednu za druhou, pokud práce nelze objektivně ohodnotit, nebo pokud se v předchozích běžích kurzu ukázalo, že velmi kompetitivní prostředí studentům nevyhovuje. Práce se neporovnává s ostatními pracemi v kurzu, ale se statickou metrikou definovanou při zadání práce. Může se jednat například o návrh ovládání a uživatelského rozhraní pro hru.

Pro konstrukci nového výukového materiálu po vzoru jednoho z uvedených typů prací, je možné vytvořit pevný harmonogram, nebo si pomoci volným prostředím pískoviště a výuku řídit osobně. Pískoviště (sandbox) označuje prostor pro volnou tvorbu, kde si uživatel může dle libosti zkoušet individuálně tvořit s minimálními vnějšími zásahy. Pro tento účel se obvykle používají nástroje třetích stran, jako Alice, GameMaker, Prog&Play nebo přímo komerční herní enginy. Přístup je vhodný pro skupinové práce nebo semestrální projekty. Pro podporu konkurenčních prací může být využit sdílený sandbox, ke kterému mají všichni studenti přístup, ať už se jedná o sdílení výpočetní síly nebo prostoru pro tvorbu.

Příkladem pískoviště je Game Maker, nástroj pro tvorbu jednoduchých her skrze grafické rozhraní a skládání bloků kódu, vhodný pro výuku programování na základních a středních školách. *Learning basic programming concepts with Game Maker* ([24]) popisuje, jak se žáci nejprve učí kód číst, teprve návazně mohou začít tvořit. Dostávají čas na rozmyšlení námětu své práce a následně jsou navedeni k vlastní tvorbě a k procvičování schopnosti řešit vlastní chyby. Učí se základy programování řízeného událostmi, stavebními prvky jsou zde objekty, které mají název, mohou mít grafický prvek, hodnoty viditelné, hmotné, mají možnost dědičnosti a také mají reakce na události a vlastní funkce. V tomto konkrétním případě se často stává, že děti nepochopí některé z prvků, a proto je nepoužívají. Později pak naráží na limitace, způsobené použitím pouze některých prvků a upravují si zadání vlastní úlohy.

Nejčastější problémy jsou spojeny se způsobem, jakým je výuka vedena. V uvedeném případě podporovala badatelské učení, tedy přístup, kdy jsou žací postaveni před nástroj a jeho dokumentaci, případně doplněnou o několik praktických ukázek, a je zcela na nich, aby se s ním naučili pracovat.

První problém je specifikace předmětu vlastní tvorby, toto není problém specifický pouze pro žáky základních a středních škol, pokud je úloha příliš otevřená, může být složité přijít s vlastním konceptem, který bude splňovat zadané podmínky. V takovém případě je lepší převzít cizí návrh a zpracovávat ten, aby se student alespoň z takového příkladu mohl něco naučit.

Dále dokumentace nástrojů není vždy ve skvělém stavu, proto může být celkem složitou úlohou si podle takové dokumentace osvojit i jen základní ovládání. Žáci měli v řadě případů problém správně pochopit jednotlivé konstrukce či koncepty, nebylo jim jasné praktické využití dostupných prvků. Formální představení základních možností a příklady jejich užití by v tomto ohledu pomohly.

3.4 Gamifikace ve výuce

Základní rozlišení vážných her, gamifikace, hravého návrhu a hraček popisuje *Publication trends in gamification: A systematic mapping study* ([25]). Hračky a vážné hry jsou obecně dobře chápané pojmy. Hračky jsou navržené pro hraní a nemají jiné využití. Vážné hry používají herní komponenty a lze je využít pro zábavu i pro jiné reálné účely.

Obtížněji se rozlišuje mezi gamifikací a hravým návrhem. Pro rozlišení se hodí věnovat pozornost termínům používaným v angličtině, tedy *gaming* a *playing*. K popsání rozdílu mezi těmito pojmy se používá tří stavů: vážný, hravý a herní. Výchozím stavem člověka je vážný, pokud se ale dostane do netradiční situace, přechází do hravého stavu (v angličtině se označuje jako playing state), což se projevuje tím, že se automaticky snaží koncentrovat na problém a udržet se naživu. Praktickým příkladem může být cyklista, který neúmyslně sjel z cesty a řítí se z prudké stráně. Je maximálně koncentrován na vlastní přežití a minimalizaci škod.

Po zkušenosti s hravým stavem a danou situací se člověku naskytuje otázka, jaké jsou hranice jeho schopností a zda by se neměl na podobnou situaci lépe připravit. V případě, že se rozhodne úmyslně situaci navštívit, přechází z hravého stavu do herního stavu (angličtina označuje jako gaming state). Je cílevědomý, používá vyšší řád komplexity pro řešení problému, připravuje se a snaží se dosahovat cíle lépe, konzistentně a s minimálním rizikem. V uvedené paralele se cyklista na sjezd svahu připraví, vybere si zvládnutelný svah a správně vybavení, se kterým situaci úmyslně čelí.

Nástroj, používající hravý návrh, má reálné využití a implementuje hravé komponenty. Zatímco nástroj s herním návrhem, tedy gamifikovaný nástroj, má reálné využití s použi-

tím herních komponent. Zásadním rozdílem je funkce těchto komponent, protože hravé komponenty nepřidávají značnou hodnotu nástroji, mají tendenci rozptylovat uživatele a vystavovat ho neobvyklým situacím. Herní komponenty se snaží nástroji hodnotu přidat, uživatel gamifikovaného nástroje tyto komponenty využívá zcela úmyslně se snahou dosáhnout cíle, který si sám definoval.

Autor ze svého průzkumu ([25]) vyřazuje publikace o vážných hrách, hravém návrhu a hračkách, bere v potaz pouze publikace zabývající se gamifikací. Shrnuje, že v polovině případů je zaměření těchto publikací ověření konceptu gamifikace a její obecná teorie. Častým tématem je také využití ve výuce, dopady a zlepšování motivace skrze gamifikaci. Statistické zpracování ukazuje, že se počet publikací na toto téma od roku 2014 snižoval a nyní se stabilizuje. Je možné se dohadovat, co je příčinou poklesu zájmu o problematiku gamifikace, ale zda byl koncept pouze nadhodnocován, nebo zda se ho nepodařilo doposud správně implementovat, nelze jednoznačně určit.

Obecný přístup ke gamifikaci výuky popisuje *Improving Play and Learning Style Adaptation in a Programming Education Game* ([31]). Rozlišuje čtyři základní typy stylů učení, studenti využívají při svém studiu jejich kombinaci, jeden z typů obvykle preferují. Úvodní dotazník má za úkol identifikovat, jaký typ osobnosti u daného studenta převládá. V průběhu výuky se sbírají data pro úpravu domněnky o dominantním typu.

Typ proaktivního studenta typicky jedná před samotným zamyšlením, mnohdy se ani takový student nechce nad tématem zamýšlet, stačí mu si ho vyzkoušet a pokračovat s dalším tématem bez prodlení. Reflektivní studenti sledují chování ostatních, převezmou od nich získané informace a na jejich základě upravují své chování a teprve poté se pouští do zadaného úkolu. Teoretici v první řadě problematiku analyzují a nastudují, než se pustí do utváření závěrů a samotné praxe, často ani nemají potřebu se do praxe pouštět. Pragmatici nejprve hledají důvod a případ užití, následně si zkušenosti osvojují v praxi, snaží se přebrat si pouze relevantní zkušenosti bez jiných zatěžujících faktů. Pro každý z těchto typů je ideální postup výuky mírně odlišný.

Výuka na školách se obvykle skládá z po sobě jdoucích fází vysvětlení problému, abstraktní popis jeho řešení, předvedení postupu na konkrétním případě a v poslední fázi se studenti pouští do praktických cvičení. Proaktivní studenti ale používají raději popis řešení a na jeho základě rovnou řeší praktickou úlohu, pokud se jim z abstraktního popisu nedaří dosáhnout cíle, využijí konkrétní postup a z něj se snaží problematiku pochopit, vysvětlení celé oblasti je pro ně relevantní až teprve po vyzkoušení praktického příkladu, relevance je zde čistě subjektivní.

Reflektivní studenti nejprve vstřebávají problematiku, následně se shání po řešení konkrétního úkolu a toto řešení si obhajují, jak podle jiných příkladů, tak podle abstraktního popisu řešení, finálně se pokusí ho aplikovat na praktickém příkladu. Teoretici zpravidla analyzují abstraktní postup řešení, než znají konkrétní problematiku, s jejím poznáváním si

sami vyhodnocují, jak se jim dané řešení zdá praktické v celém rozsahu problematiky a jak je efektivní, následně mají tendenci shlédnout konkrétní návod a aplikovat ho na příkladu.

Pragmatici se nejprve podívají na konkrétní řešení, poté řeší detailní problematiku, proč vlastně by se mělo používat takové řešení a po jeho akceptaci se pravidlo snaží zobecnit a aplikovat. Veškeré tyto přístupy dávají jistý smysl, ale ani jeden z nich není shodný s výše popsaným postupem výuky. Proto je možné cílit na flexibilnější výukové materiály, pro vyplnění této mezery.

Některé online kurzy programování již implementovaly formu rady na požádání, obvykle v provedení konkrétním návodem k řešení daného problému, a student si může buď příklad vyzkoušet samostatně bez návodu, inspirovat se jím, nebo se na něj podívat předem a řešit úlohu krok za krokem.

Praktické příklady by taky mohly být v jisté jednoduché formě dostupné předem pro studenty k vyzkoušení před samotným výkladem nebo v jeho průběhu. V podstatě všechny složky výuky by mohly být dostupné studentovi na vyžádání, taková cvičení se blíží spíše samostatné práci než řízené výuce, ale dovolují studentům přeskakovat důležité kroky a tím minout cíle výuky.

Adaptivní styl učení je popsán a vyzkoušen ([31]) na žácích šestých tříd základních škol a zaobaluje vyučovanou látku do počítačové hry. Proto doprovodně používá i rozdělení typů hráčů. Studenti se snaží za pomoci programování vyřešit jednoduché úlohy v programu a cílem je udržovat jejich pozornost a dát jim možnost nalezení vlastní cesty k dosažení cíle. Jejich cesty se mohou lišit pouze v drobnostech, klíčové elementy jsou pro všechna řešení společná.

Nejobvyklejším typem hráče bývá akční, to je hráč, který dosahuje svého cíle nejrychlejší cestou, zbytečně nezkoumá své možnosti, pokud je jeho výchozí možností ničit překážky, bude překážky ničit. Další typ hráče je sociální, který hru hraje primárně pro interakci, nejen s ostatními hráči, ale i s naprogramovanými postavami a předměty. Pokud se tento hráč setká s překážkou, kterou může kromě zničení také odsunout, nejspíš si vybere metodu odsunutí.

Hráč badatel bude program zkoumat, hledat jeho omezení, chyby a skryté části, bude využívat jiné mechaniky k obejití překážky, jenom proto, že je to možné. Poslední hráč sběratel se snaží ve hře dosáhnout všeho. Pokud lze sbírat předměty, bude shánět veškeré předměty, přestože je nikdy nevyužije, bude také zkoumat rozsah celého prostředí a pokud hra obsahuje úspěchy za plnění vedlejších úkolů, bude plnit tyto úkoly pro odemknutí medailí za dané úkoly.

Studie ([31]) proto navrhuje nabídnout studentům více řešení, případně průběžně skrze aplikaci identifikovat jakým typem hráče jsou a přizpůsobovat aplikaci jejich zájmům. Akčního hráče poznává podle zabitých nepřátel a zvyšuje pro něj počty nepřátel. Sociálního hráče poznává podle počtu interakcí s objekty, pro něj implementuje do hry volitelné

dialogy a možnosti. Badatelům je dobré do programu přidat skryté místnosti a záměrné chyby programu, bohužel nepřichází s formou identifikace badatelů a sběratelů. Sběratelům rozšiřuje hru o předměty, které nemusí nutně přinášet hře něco nového, pouze zvyšují její diverzitu, dává jim větší šanci na získání těchto předmětů, a navíc pro ně implementuje dosažené skóre.

Zde ([31]) se adaptibilita aplikace používá k získání a udržení pozornosti studenta po delší dobu. Při výuce programování je také možné studentům nabídnout možnost volby některých cvičení nebo jejich variací podle jejich vlastních zájmů. Akční hráč by mohl implementovat stavy živý a mrtvý, sociální hráč odemčený a zamčený.

Z principu výuky uvažované touto diplomovou prací, kde se používá jedna stále rozšiřovaná aplikace je tento koncept použitelný. Student může implementovat řešení, které ho láká nejvíce, v příští lekci může pokračovat s verzí aplikace, která již všechny nabízené možnosti bude obsahovat.

3.4.1 Gamifikace s využitím osobnostních typů

Kapitola vychází z *Profile-Based Algorithm for Personalized Gamification in Computer-Supported Collaborative Learning Environments* ([26]). Představuje prototyp výuky, která je zatím stále iterativně upravována. Cílem gamifikace v tomto případě je zvýšení spolupráce studentů a dosažení více pozitivního přístupu k výuce. Rozlišuje typy učení: individuální, skupinové, kolaborativní a hromadné. Individuálním učením se zde rozumí situace, kdy na vyučujícího připadá velmi málo studentů, kterým se individuálně věnuje a tito studenti se navzájem neovlivňují. Skupinové učení popisuje stav, kdy jsou studenti rozděleni do týmů a každý tým pracuje na společném úkolu. Kolaborativní učení zadává studentům stejné nebo velmi podobné úkoly, studenti mají stejné role a podobné znalosti, mohou si ale libovolně pomáhat s řešením úkolů. Příkladem hromadného učení je běžná přednáška, kde orientace na individuální studenty je téměř nemožná. Kolaborativní učení je doporučeno a využíváno ve výuce popisovaného prototypu.

Pro účel personalizace adoptuje a popisuje typy herní osobnosti. Filantrop, altruistický, motivovaný účelem a ochotný pomáhat ostatním. Společenské osobnosti jsou motivovány komunikací, sdílením poznatků, dojmů a nápadů. Nezávislí jsou motivováni možností vlastní volby a zkoumáním. Dosažitelé chtějí dokázat sobě nebo i okolí, že zvládají složitější úkoly, než jaké jsou před ně postaveny. Osobnost hráče se chce prosadit na vrchol tabulek, získávat nejvyšší hodnocení, dosáhnout ocenění či cen.

V přípravné fázi kurzu proběhla diskuse s experty na návrh her, gamifikaci a softwarové inženýrství. Z té vyplynuly základy frameworku pro sledování chování studentů, předpoklady aplikace personalizované gamifikace a pravidla návrhu gamifikovaných prvků. Aktivita uživatelů se sleduje a je chápána jako řetězce chování, skládajících se z akcí mířených na plnění potřeb kurzu.

Předpokládá se, že studenti mají zadané vlastní úkoly a jsou ochotni a mohou si navzájem pomáhat. Že existuje systém na kontrolu stavu zadaných úkolů, který umožňuje zjistit, na kterém úkolu student pracuje, jak dlouho na něm pracuje a také umožňuje mu poskytnout externí pomoc. Také předpokládá existenci a použití nástroje volné komunikace, ideálně s možností vytvoření většího počtu tematických komunikačních kanálů, historií konverzací a možností přímé privátní komunikace pro maximalizaci úspěchu a využitelnosti přenosu informací mezi studenty.

Pravidla, která dodržuje při návrhu systému pro poskytování personalizované gamifikace jsou, že je potřeba používat herní návrh (nikoli hravý návrh) jednotlivých prvků. Studenti využívají tyto prvky dobrovolně a mohou se jim kdykoli vyhnout. Opakované použití stejného generického prvku není příliš efektivní a může snadno omrzet. Množství prvků gamifikace by mělo být rozumné, ve vztahu k počtu studentů a délce kurzu. Je třeba nalézt optimální množství prvků, jelikož málo prvků není dostatečně zajímavých a mnoho prvků by mohlo být spíše matoucí. Studentům je třeba nastavit výzvy, které jsou náročné, ale zvládnutelné, ideálně na hranicích jejich schopností a časových možností. Pomáhá poskytovat pozitivní zpětnou vazbu v případech, kdy student něčeho dosáhl. Interakce mezi studenty by měla být podporována a usnadňována.

Prvky gamifikace by neměly odvádět pozornost studentů od úkolu. Vlastní potřeby studenta by měly být podporovány, ale nikoli na úkor jiných potřeb nebo potřeb kurzu, tedy pokud je student motivován dosažením vysokého hodnocení, neměl by být oceňován za plnění mnoha vedlejších úkolů, nijak nesouvisejících s cíli výuky. Neměl by také potlačovat složku komunikace s ostatními za účelem dosažení svého osobního cíle. Systém by měl být otevřený a pozitivní, jeho funkce ve výuce má být podpůrná, nikoli klíčová. Ve finále by systém měl být dostatečně flexibilní, rozpoznávat uživatele a navrhovat vhodné úkoly. Nelze předpokládat, že uživatel má vyhraněnou osobnost, jeho přístup se může a měl by se v průběhu měnit.

Pro inspiraci uvádí v praxi ověřené typy úkolů s konkrétními příklady pro každou z osobností. Filantrop může dostávat úkoly, aby pomohl jiným. Například pokud je daný student zkušený v dané problematice a ve skupině jsou studenti, kteří mají s řešením úkolu problémy, může daný filantrop dostat za úkol napsat do chatu a získat kladné hodnocení od ostatních studentů na daný příspěvek. Společenský člověk může podporovat komunikaci v kolektivu. Méně zdatný sociální student v prostředí, kde jsou zdatnější studenti, může dostat za úkol účastnit se aktivní konverzace v chatu po dobu několika desítek minut.

Nezávislí rádi následují úkoly zkoumání, bádání, sdílení informací a objevování. Pokud jsou součástí relativně neaktivního týmu, rádi se ujmou úkolu zahájit diskusi v chatu pro zmapování pokroku ostatních týmů. Dosažitelé mají rádi milníky, výzvy a odměny. Pokud mají například týmy vyrovnané skóre, ochotně se snaží získat navrch oproti ostatním. Hráči jsou z pohledu motivace velmi podobní dosažitelům, sledují dosažení milníků, vyšších úrovní, odměn nebo vyšší efektivity. Jedním z úspěšně aplikovaných úkolů je

řešení problémů, pokud tým nestíhá, ať už vlastními silami nebo získáváním pomoci od ostatních.

Ačkoli autor uvádí, že týmové úkoly nejsou podstatnými pro personalizovanou gamifikaci, praktické příklady jejich použití zřetelně implikují. Vytyčená pravidla by se dozajista dala úspěšně aplikovat na běžnou výuku. Z počátku by bylo rozumné ověřit koncept na menším vzorku studentů bez implementace složitějších systémů, a to přímým zásahem vyučujícího.

3.4.2 Gamifikace napříč kurzem

Zde se shrnují poznatky studie *Climbing Up the Leaderboard: An Empirical Study of Applying Gamification Techniques to a Computer Programming Class* ([38]), která se zabývá gamifikací celého kurzu. Tedy přechodu od tradičního přístupu, skládajícího se z přednášek, praktických cvičení a interaktivních seminářů určených pro studenty, na výuku skrze hry. Problémy tradičního přístupu byla nízká a pozdní docházka, využívání studijních materiálů zejména v prvním týdnu výuky a poté až před závěrečným testem, nedokončování zadaných úloh a špatná úroveň výsledků testů. Proto byly přednášky rozšířeny o online kvízy v reálném čase, kde studenti zodpovídají otázky a vyučující si tím může ověřit, zda byla látka správně pochopena, případně dovysvětlit nejasnosti a zmatky. Praktická cvičení byla přesunuta do online prostředí CodeAcademy – virtual classroom, kde studenti plnili zadané úlohy, odemykali dosažené úspěchy a mohli sledovat jejich vlastní postup oproti svým spolužákům. Interaktivní seminář změněn na hru typu „Chcete být milionářem?“, studenti rozdělení do skupin se postupně střídali v roli soutěžících, získávali body za správné odpovědi a tyto body se jim kumulovaly napříč semestrem. Bodování studentů bylo přístupně pro ostatní studenty, aby byli účastníci kurzů motivovanější při plnění zadaných úkolů. Studentům byly zpřístupňovány krátkodobé i dlouhodobé úspěchy, ve formě odměn za nejlepší bodové ohodnocení z daného dne, týdne nebo semestru.

Pozorování ukázalo, že studenti gamifikované výuky projevovali aktivnější účast na rozdíl od předchozích běhů. Studenti projevovali vyšší pozornost, drželi oční kontakt, okamžitě reagovali na instrukce pedagogů a aktivně se rozhodovali, pokud byl předložen otevřený návrh. Ve skupinových úlohách se snažili více a lépe komunikovat a řešit důležité problémy. Celkově řeč těla studentů prozrazovala otevřený a uvolněný přístup, který nebyl obvyklý při předchozí tradiční výuce. Ze zpětné vazby vyplývá, že studentům nevadilo zveřejnění jejich statistik. Nepotvrdilo se riziko, že by se někteří studenti rozhodli se podobných praktik vůbec neúčastnit, kdyby měli dojem, že je na ně příliš velký společenský tlak a že získání dostatečně dobrých výsledků je pro ně příliš obtížné. Finálně se ze statistik stahovaných materiálů ukazuje, že studenti gamifikovaných kurzů stahovali studijní materiály průběžně, bez výkyvů před testy.

Jistá forma gamifikace by se na základě této studie převzít i do běžných kurzů programování bez zásadních změn výuky. Prvek vyhodnocování efektivity předávání informací studentům na přednáškách jednoznačně stojí za zvážení, největší protiváhou je zde praktič-

nost a časová náročnost daného řešení, studium na vysoké škole spoléhá na samostudium jako formu studia doplňující právě látku, která se nestačila probrat v hodinách, a proto jsou podobné metody většinou aplikovány ve skupinách na cvičeních nebo na menších seminářích. Řada přednášek se snaží o aktivní zapojení studentů do konverzace, obvykle se to daří pouze v několika případech, a to jen tehdy, pokud je přítomen dostatek studentů s vysokým zájmem a rozhledem v probírané problematice. Gamifikovaný přístup neovlivňující výsledné hodnocení studenta by mohl zvýšit aktivitu dané přednášky bez negativních vedlejších dopadů na výuku. V samotných cvičeních programování, i ve stavu v jakém jsou dnes, by šlo velmi jednoduše nastavit sadu jednotkových testů, které by ověřovaly snažení studentů z více stránek, každý test by měl přiřazené bodové ohodnocení, student by získal body, pokud by daný test prošel. Získání vysokého skóre by vedlo k získání jistých benefitů nebo ocenění, zde velmi záleží na povaze kurzu a studentů, aby se taková praktika neminula účinkem.

3.4.3 Skriptovací prostředí pro podporu výuky

Běžně se využívají k definici black-box testů spojených s vyhodnocením splnění úkolu studenta. Může být definována řada skriptů, které se spouští během, nebo závěrem výuky, ať už interně v rámci vylepšovaného programu, vývojového prostředí, kolaborativního systému nebo externě. Obvykle tyto skripty zajišťují zpětnou vazbu vyučujícímu nebo studentům a spouští se na vyžádání, proaktivně nebo reaktivně. Tímto způsobem lze studentům poskytovat cílenou nápovědu. V případě gamifikace výuky, mohou tyto skripty zpracovávat informace o akcích, událostech, průběhu a historii průchodu hrou studentem.

Computational Tools for Computing Education ([54]) definuje a kategorizuje nástroje pro podporu výuky programování. Věnuje se výhradně nástrojům podstatným pro výuku, nikoli doplňkovým. První popsanou kategorií jsou kontextuální nástroje, které se používají v konkrétních situacích a nejsou potřeba po celý průběh kurzu. Mezi takové nástroje řadí automatické vyhodnocování programovacích úkolů nebo detekce plagiarismů.

Druhou kategorií jsou zde nástroje pro skupinu uživatelů. Na rozdíl od první skupiny, kterou nejčastěji využívá vyučující, tyto nástroje využívají zejména studenti. Řadí sem zejména vývojová prostředí podporující výuku, jako je Alice, Scratch nebo Greenfoot.

Poslední kategorii definuje jako nástroje zaměřené na uživatele. K této kategorii autor uvádí, že nenalezl reálný příklad. Popisuje alespoň stručně očekávané vlastnosti a přínosy. Nástroj by měl uživateli pomáhat řešit konkrétní problémy a měl by být iterativně vylepšován na základě zpětné vazby.

Bohužel neuvádí více příkladů, ani se nezabývá jejich implementací a užitím. Označení účelu nástroje, jako podpora výuky, se zdá velmi zjednodušená. Bylo by vhodné se zaměřit na přínosy jednotlivých nástrojů, pokud by měly být využívány ve výuce. Příklady uvedené autorem snižují náročnost úkolů studentům a jejich kontroly vyučujícím. Častým téma-

tem bývají ale také nástroje pro zvýšení motivace. Je také možné, že je autor nepovažuje za stěžejní, a proto se takovým nástrojům nevěnuje.

Scripting Environments of Gamified Learning Management Systems for Programming Education ([52]) se zabývá otázkou výhod a nevýhod spouštění skriptů na serveru nebo u uživatele. Použitelnost serveru je závislá na množství studentů, kapacity serveru může být třeba upravit. Při exekuci na straně klienta tato starost odpadá, přibývá starost, zda není skript příliš náročný na hardware. Responzivita serveru je běžně nižší, pokud jsou ale procesy sdílení dat dobře nastaveny, rozdíly jsou v rámci běžného užití minimální.

Pro použití klíčových vyhodnocovacích skriptů na straně serveru mluví manipulovatelnost výsledků, která je výrazně snazší v případě skriptů na straně klienta. Ovšem takové skripty lze spouštět lokálně po odevzdání práce. Pokud je cílem zvýšit zapojení studentů do výuky, může pomoci stylizovat cvičení jako soutěž, což se lépe provádí s využitím serveru a vyhodnocování v reálném čase.

Využívání různých skriptů a nástrojů pro podporu výuky dozajista stojí za zvážení, je třeba ale brát v potaz náklady na jejich zavedení. Zejména náklady na zajištění serveru, úpravu cvičení a implementace gamifikace se mohou ukázat příliš vysoké.

3.4.4 Návrh vážné hry

Kapitola shrnuje poznatky o návrhu vážné hry podle *Game design for a serious game to help learn programming* ([45]). Hra by se měla skládat z příběhu, umělecké složky a ze softwaru, pro vážné hry je potom dále důležité implementovat jisté pedagogické prvky.

Jedním z prvních identifikovaných problémů výukových her je zde malý rozpočet na vývoj. Autor zastává názor, že by se tyto hry měly blížit kvalitou komerčním hrám. Zde by bylo vhodné rozlišovat hry na komerční, komerční výukové a čistě výukové, kde komerční hrou se rozumí software vytvořený za účelem dosažení zisku, komerční výukové hry jsou softwarem vytvořeným za účelem přilákání účastníků do placených kurzů a čistě výukové hry jsou tvořeny obvykle pedagogy jako podpůrný studijní materiál s cílem zvýšit atraktivitu výuky pro stávající studenty.

Komerční výukové hry mohou být neúspěšné, pokud nedosahují požadované úrovně kvality z důvodu nedostatečného rozpočtu nebo uspěchaného vývoje. Čistě výukové hry obvykle nemají přiřazený rozpočet a jejich kvalita je ve určena časem vloženým do jejího vývoje. Autor této práce neřeší, jak by čistě výuková hra měla dosáhnout požadované úrovně. Běžně tato činnost spadá na vyučující programování, případně ve spolupráci se studenty.

V univerzitním prostředí by bylo možné využít souvisejících předmětů z oblasti návrhu softwaru, grafiky, databází nebo kvality software. Takové propojení by jednak poskytovalo zajímavý praktický příklad na procvičení a zároveň by vytvářelo nové studijní materiály.

Další výhrady autora k výukovým hrám se váží na propojení hry a vyučované látky. Řada her se neřídí svými základními principy, ani cíli výuky. Často se projevují domněnky atraktivního prvku jako samostatný prvek kompletně neovlivněný řešenou problematikou. V aplikaci jsou potom pro studenta předloženy dva izolované úkoly namísto jednoho, jeden z úkolů zde řeší příběh nebo princip hry, druhý představuje alternativní možnost řešení. Nejčastěji se takto projevují hry pro výuku kódování na základních a středních školách, kde má student naskriptovat pohyb figurky skrz hrací plán, student zde využije sérii příkazů namísto cyklu, které je až následně představen jako alternativní řešení.

Takové hry obvykle mívají další nepříjemnou vlastnost. Trénink a opakování převažuje nad snahou o porozumění. Pokud se student již jednu strukturu naučil, měl by jí být schopen používat ve více situacích, přesto není nezbytné studenta nutit tuto strukturu vytvářet stále dokola. Daleko efektivnější je strukturu znova použít, nebo ji studentům předpřipravit, aby se zbytečně neplýtval čas na dlouhé řadě jednoduchých úkolů.

Výukovým hrám se dále vyčítá, že jsou příliš jednoduché. Na jednu stranu příliš jednoduché hry často omezují zavedení komplexnějších konceptů a podporují repetitivní implementaci velmi podobných struktur, na druhou stranu složitější hry mohou být pro studenty těžší na uchopení. Složitá hra, která je z velké části již hotová a studenti ji pouze upravují, bude často potřebovat zavést interní framework.

Posledním z problémů, na které autor upozorňuje, je snaha aplikace učit se samostatně bez využití učitele. Osobně bych to neoznačoval za problém, spíše za rozhodnutí závislé na kontextu výuky. Ve prospěch samostatně vyučující aplikace mluví rozdílnost studentů, pokud to způsob výuky podporuje, studenti se mohou probrat takovým programem vlastním tempem, například formou domácího cvičení. Výuka na cvičení se obvykle snaží držet všechny studenty na stejné kapitole a v takovém případě samostatnost aplikace, může představovat jistou nevýhodu.

Jako příklad dobrého návrhu se zde uvádí otevřená kompetitivní hra. Soutěživost je dozajista dobrým motivátorem, ale dopad takového návrhu je potřeba předvídat a řídit. V naprosté většině případů výukových kompetitivních her vězí podstata soutěže ve schopnostech rozhodovacích algoritmů, jejich rychlosti a správnosti. Což při výuce návrhových vzorů nepřináší žádnou výhodu, takový prvek právě naopak bude mít tendence odklonit pozornost studenta od dobrého návrhu k promyšlenosti konkrétního algoritmu.

Známkou dobrého návrhu je také možnost volby a rozhodnutí studenta. Možnost volby je celkem širokým pojmem. Může se jednat o volbu cíle, kterého se snaží student dosáhnout. Obecně atraktivnější výukové hry nabízejí studentům ke splnění více možných úkolů, jejichž podstata je stejná. Také si student může vybírat způsob jakým konkrétního cíle dosáhne. Nejužitečnější možností volby je obecně rozhodnutí, zda řešení bude univerzální, nebo pouze minimální možné. V případě, že se student rozhodne pro jednodušší minimální řešení, může mu být obratem rozšířeno zadání, aby si vyzkoušel náročnost spojenou s rozšířením méně ideálního návrhu.

Dobře navržené hry také kladou studentům překážky, tyto překážky můžeme klasifikovat na kritické, velké, malé a triviální. Kritické překážky student musí vyřešit, aby splnil zadání. Velké překážky by měl samostatně identifikovat jako chybu vlastního díla a může se rozhodnout je samostatně opravit. Malé překážky bývají spíše nastraženy pro studenty, aby se na nich s jistou moderací ze strany pedagoga mohli naučit něco navíc. Pedagog je také může využít k předvedení kvality použitého návrhu. Triviální překážky jsou spíše volitelnou sadou vylepšení pro aktivní studenty.

Hra by se měla držet zavedených zásad, není dobrým návrhem poskytnout jednu metodu na řešení problému, jen aby byla později nahrazena lepší podobně komplexní metodou. Tyto zásady by měly vycházet z kontextu výuky. Jiné zásady adaptuje výuková hra podporující výuku konkrétního programovacího jazyka než hra pro výuku návrhových vzorů.

Hlavním požadavkem, který práce definuje pro návrh vážné hry, je, že hlavní mechanikou by měla být úprava a tvorba kódu, nikoli interakce s ovládacími nebo vestavěnými řídícími prvky. Dalším požadavkem je ale jazyková nezávislost, čímž nepřímo požaduje tvorbu vnitřního pseudokódu a jeho překladače, jinak by dosažení vysoké kvality hry a vlastní úpravy kódu byla časově nezvladatelná pro účely klasické výuky.

Výsledný produkt by měl používat populární atraktivní mechaniky komerčních her pro podporu výuky a zároveň by měl neměl obsahovat přebytečné vlastnosti. Jsou zde ale otázky, ke kterým autor nezaujímá jednoznačný postoj, například užití 3D grafiky nebo 2D. Zatímco 3D je vizuálně atraktivnější, 2D je snazší na uchopení pro začátečníky. Tento problém už řada komerčních her vyřešila, používají hru v základě 2D, ale přidávají do ní grafické elementy vzbuzující dojem 3D prostředí.

Z pohledu přístupu k výuce práce požaduje, aby návrh podporoval metodu pokus omyl. Řada předmětů staví studenty do testových situací, kde jednoznačně rozhoduje, zda student ví nebo neví. Tento přístup je v oblasti programování minimální v porovnání například s účetnictvím, už jenom z podstaty programování. V neposlední řadě chce propojit výukovou hru s jinými problematikami, aby si studenti uvědomili rozsah aplikace, tedy s grafickým návrhem, užitím databází nebo síťové komunikace.

Návrh vážné hry by se měl inspirovat návrhem komerční hry, proto definuje seznam prvků, které by návrh vážné hry měl řešit:

1. Koncept hry – žánr, popis, hlavní vlastnosti
2. Mechaniky, funkční elementy
3. Uživatelské rozhraní (UI/UX)
4. Grafický koncept hry
5. Příběh a postavy
6. Požadavky úrovní – jejich počet a návaznosti
7. Vývojové prostředí

8. Architektura, framework
9. Umělá inteligence
10. Propojení hry pro více hráčů
11. Konkrétní návrh jednotlivých úrovní

S autorem nelze jednoznačně souhlasit v definovaných prvcích návrhu vážné hry. Přinejmenším je vhodné nastavit prioritu jednotlivých prvků, řada z nich by totiž spadala do kategorie věcí, které by bylo pěkné zohlednit, ale nejsou zásadními pro finální produkt. Je třeba začít konceptem hry, požadavky na úrovně, architekturou a vývojovým prostředím. Poté navrhnout jednotlivé úrovně včetně používaných mechanik a elementů.

Grafický koncept hry, návrh jednotlivých úrovní a uživatelského rozhraní by mohl být řešen formou projektu z předmětů se zaměřením na počítačovou grafiku. Příběh, postavy a umělou inteligenci jsou spíše vlastnosti navíc, řada úspěšných her dokazuje, že atraktivní jsou i samotné mechaniky hry bez příběhu.

3.4.5 Vyhodnocení kvality výukových her

Tématu se věnuje *State of the art in Game Based Learning: Dimensions for Evaluating Educational Games* ([53]). Ukazuje, že zastoupení tématu vyhodnocování výukových aktivit ve spojení s gamifikací v odborných publikacích obecně dlouhodobě roste, zvláště výrazné zastoupení bylo patrné mezi roky 2009 a 2015, ale v posledních dvou letech mírně klesá. Nejčastější kritéria vyhodnocení se týkají výuky, často se hodnotí použitelnost, návrh a použité mechaniky. Vyhodnocování kvality a použitelnosti her ve výuce podle většiny uvedených metodik dává smysl zejména mezi cykly výuky při jejich iterativní úpravě. Mohou být nejlépe využitelné jako podklady a inspirace pro zlepšení.

Analýza vyhodnocovacích přístupů je velmi stručná, autor ([53]) se zejména odkazuje na literaturu popisující dané přístupy, nerozebírá již jejich podstatu ani zastoupení ve zkoumané literatuře. Nejstarším popsaným přístupem je zde čtyřdimenzionální framework pro vyhodnocování praktického potenciálu her a simulaci pro výuku. Další často adoptovaný přístup byla metodika vyhodnocování o porovnávání her, zaměřená na kvalitu a vhodnost příkladu. Používají se také metodiky pro vyhodnocování efektivity použití her pro výuky z pedagogického hlediska, metodiky pro vyhodnocování plynulosti her v online výuce, rozšířená metodika vyhodnocování použitelnosti a vhodnosti konkrétních konstruktů při návrhu výukových her a pro vyhodnocování vhodnosti uživatelského rozhraní aplikace.

Používají se ([53]) i heuristické modely vyhodnocení jak z pedagogického pohledu, tak z pohledu aplikovatelnosti na téma a spokojenosti studentů. Využívají se různé příručky pro vyhodnocování, definují se návrhy na standardy a modely vyhodnocení kvality her a výukových her. Speciální metodiky jsou používány na vážné hry, opět jsou různé metodiky na různé aspekty těchto her.

Vážná hra v případové studii *Jeu sérieux et motivation des étudiants pour apprendre: influence du contexte avec Prog&Play* ([35]) využívá námět strategické hry v reálném čase, úkolem studenta je naskriptovat chování jednotek pro splnění misí a následně pro souboje mezi studenty. Předpoklady aplikace této hry do výuky jsou, že studenti jsou motivování příběhem nebo soutěživostí plnit úkoly, mají na tuto aktivitu vyhrazených zhruba 8 hodin výuky, chtějí vidět dosažené výsledky a forma a cíle výuky povolují výhradní použití předpřipravené knihovny pro skriptování úloh.

Byla měřena subjektivní spokojenost studentů ([35]) motivovaných k aktivitě studijními povinnostmi a vlastním rozvojem ve vstupních kurzech programování. Z měření vyplývá, že studenti, kteří absolvovali toto cvičení povinně, byli s touto aktivitou nespokojeni. Ale ani u studentů, kteří se aktivity účastnili dobrovolně, nebyla příliš vysoká spokojenost, stále většina studentů hodnotila zkušenost spíše negativně. Pro porovnání byla aktivita předložena i studentům navazujících kurzů, jejichž spokojenost byla výrazně vyšší. Úloha se proto jeví příliš složitá pro studenty vstupních kurzů, je vhodné ji buď zjednodušit, nebo přesunout do návazných kurzů.

Na stejnou výukovou hru a skupinu studentů navazuje *Étude de l'intégration d'un jeu sérieux pour l'enseignement de la programmation dans différents contextes universitaires* ([34]). Identifikuje, že atraktivnost hry nespočívá pouze v její podstatě, ale také na přidané hodnotě z hlediska použití v praxi. Autor zde uvažuje pouze dva modely výuky, kde student buď hru sám programuje, nebo pouze skriptuje její chování. Je pochopitelné, že se snaží využít existující volně dostupné knihovny, ale bylo by bezpochyby možné poskytnout studentům framework pro programování hry, pro případy výuky, která nemá za cíl učit skriptování.

Studentům po absolvování kurzu předkládá dotazník, vybrané odpovědi jsou uvedeny přímo ve studii ([34]). Z průzkumu vyplývá, že studenti silně pozitivně vnímají zábavnost hry, aplikovatelnost látky v praxi, vytvořený prostor pro kreativitu a spolupráci, inovativnost přístupu, komplexitu problému a dopad na výuku. S podporou aktivity ze strany pedagogů nemají problémy, ale ani ji nevnímají jako výrazně propracovanou a užitečnou. Negativně na studenty působí použitý programovací jazyk (výuka je připravena pro více programovacích jazyků, problémové jsou případy užití C a Caml) a časová náročnost cvičení. Nejvýrazněji problematická byla organizace cvičení a náročnost její přípravy ze strany pedagogů.

Na základě zpětné vazby ([34]) byla sepsána sada doporučení pro implementaci nástrojů podobných Prog&Play. Vyplývá z ní, že by se pedagog měl detailněji seznámit s příkladem, podle kterého vyučuje a s jeho náročností. Vyhodnocení aktivity by se mělo přizpůsobit znalostem a schopnostem studentů. Cíle výuky by se měly na začátku jasně stanovit, student by měl mít dostatek času prozkoumávat aplikaci a mělo by mu být vysvětleno proč a jak má využívat knihovny. Pomáhat úspěchu této aktivity může pedagog podpořením spolupráce studentů a nalezením vhodného způsobu podpory studentů s rozdílnou úrovní schopností. Na závěr je vhodné rekapitulovat přínosy aktivity. Popsaný průzkum lze repli-

kovat a použít na analýzu jakéhokoli výukového materiálu. Sekce doporučení z této studie je aplikovatelná na libovolné praktické cvičení.

V jiném případě se vyhodnocením příkladu vážné hry zabývala případová studie *Analysing the Enjoyment of a Serious Game for Programming Learning With two Unrelated Higher Education Audiences* ([55]). Zde se uvažuje aplikace s principy a elementy běžné hry, ale se základem a inspirací v běžných problémech, často se takové vážné hry prolínají se simulacemi. V tomto kurzu se používá jako nástroj pro programování Scratch a v něm vytvořená hra No Bug's Snack Bara, kde hráč programuje chování číšníka v restauraci, aby dostatečně rychle obsloužil zákazníky.

Základními požadavky na takovou hru ([55]) byly intuitivnost ovládání, zvyšující se obtížnost a fantasy elementy pro oživení. Připravená hra byla vyzkoušena na skupině studentů a jejich preference ukázaly, že dávají přednost hrám pro jednoho hráče s příběhem. Tato preference obecně vyhovuje vážným hrám, jelikož ty nejsou obvykle koncipovány pro hru více hráčů. V tomto ohledu se dají hráči kategorizovat buď mezi příznivce samostatné hry s příběhem, příjemným zvukem a grafikou, nebo mezi příznivce hry více hráčů, kde na grafické věrnosti nebo příběhu tak nezáleží, hlavní je zkušenost ze hry proti ostatním hráčům, která je i na menším herním obsahu bohatší, díky rozmanitosti přístupů jednotlivých účastníků.

Zpětná vazba a vyhodnocení kurzu ([55]) ukázala, že by studenti uvítali element nejistoty, například aby po mnohonásobném selhání následovala penalizace. Samotná hra by měla obsahovat instrukce k jejímu ovládání a postupu, měla by mít bohatší příběh, audio i grafiku, což vychází z jejich oblíbených her, které toto implementují jako standard. Atraktivitu by to nejspíš pro tyto studenty zvýšilo, není ale jasné, zda by to přineslo nějakou přidanou hodnotu pro samotnou výuku. Posledním požadavkem bylo, aby se obtížnost zvyšovala, se zachováním časové náročnosti jednotlivých úloh.

3.5 Pedagogické vzory

Pedagogové se zkušenostmi s výukou programování se inspirovali návrhovými vzory programování a začali si definicí vlastních vzorů pro výuku. *Pedagogical patterns: advice for educators* ([3]) sumarizuje řadu vzorů, které lze považovat za výchozí a ověřené, jejich výhody a nevýhody. Na toto téma začalo reagovat více pedagogů a vytvořili řadu ekvivalentních vzorů.

Proto je vhodné se věnovat i publikacím, které tyto vzory sumarizují a třídí, například *Lecture Design Patterns: Laying the Foundation* ([27]). Definuje hlavní obecné zásady, které zahrnují výběr vhodného obsahu z hlediska obtížnosti, zaměření a přidané hodnoty. Výběr vhodné formy a média předávání informací. Obvykle je dobré kombinovat použití skript, výkladu, diskusí, cvičení, ukázek a workshopů. Není vhodné se dlouho držet jedno-

ho způsobu výuky, jednoho média. Změna přístupu pomáhá zpestřit výuku a udržet zvýšenou úroveň pozornosti. Látku je třeba rozdělit na části, souvislý celek je obtížně uchopitelný a vstřebatelný. Stimulace představivosti skrze praktické příklady nebo kreativní úkoly pomáhá studentům problematiku lépe pochopit.

Autor ([27]) identifikuje související jednodušší výukové vzory, zásady a praktiky, ze kterých se výše zmíněné skládají. Z pohledu přípravy kurzu je vhodné připravit detailní studijní plán, specifikovat a zkontrolovat požadavky. Pokud to může kurzu prospět, případné hry, simulace nebo workshopy by měly být připraveny předem. Vyplatí se průběžně identifikovat a předcházet různým komplikacím, které mohou nastat. Pokud kurz přináší podobnou látku, je dobré tyto podobné sekce oddělit, aby se studentům nepletly. Stejně jako u libovolného plánování, je dobré držet si rezervy a najít uplatnění pro případné prostoje vzniklé nevyužitím rezerv.

Z pohledu samotné výuky je vhodné představit na začátku přehled problematiky. Používat relevantní příklady a obeznámit se s nimi. K lepšímu pochopení často pomáhá využití metafor, uvedení stejného problému v jiném kontextu nebo příkladu. Nejprve představit koncept, vysvětlit proces, postupně se propracovat přes širší perspektivu až k úplné abstrakci a následně abstraktní koncept shrnout, případně představit ve více implementacích. Pokud je potřeba představit složitější koncept, je dobré ho nastínit s předstihem a vybrat vhodnou úroveň detailu k jeho popsání a chápání.

Pro smysluplné učení je vhodné reaktivovat osvojené znalosti například prostřednictvím jednoduchých otázek. Následně známé rozšířit o nové. Připravit si prostředí a klást návodné otázky, nasměrovat studenty jistým směrem. Ideální je se orientovat na problémy, nikoli na znalosti. Je dobré používat komentované ukázky, je lepší používat médií a slovníku účastníků, pokud to nekoliduje s účelem kurzu. K identifikaci vhodných médií lze využít bezprostřední zpětné vazby.

V závěru hodiny je dobré zodpovědět otázky, na jejich základě lze upravit obsah příští lekce. Shrnout, zopakovat přínos hodiny. Vždy je dobré být připraven opustit plán a přizpůsobit se.

Konkrétní výukové vzory sumarizuje *Towards a Pattern Language for Lecture Design: An inventory and categorization of existing lecture-relevant patterns* ([28]) na základě *Pedagogical patterns: advice for educators* ([3]) a rozšiřuje je o další výukové vzory. Jejich do češtiny přeložená verze zbavená duplicitních záznamů je připojena k této diplomové práci jako Příloha B: .

V první řadě rozděluje ([28]) výukové vzory podle toho, na jaké úrovni výuky je vzor používán, používá označení Level of Education Action. Vzory přímé interakce (LEA1), jsou elementární a používají přímé interakce, trvají v řádech minut. Situační vzory, Vzory chování v průběhu lekce (LEA2), popisují vzory chování aplikované na jednu výukovou lekci. Dále rozlišuje tematické vzory, Vzory výuky problematiky (LEA3), které jsou ob-

vykle aplikovány při výuce jednoho celku napříč více lekcemi. Posledními jsou studijní vzory, Vzory vedení kurzu (LEA4), které je třeba udržovat alespoň po dobu trvání kurzu.

Další dělení ([28]) zohledňuje velikost skupiny, kategorizace podle Group Size. GS1 označuje vzory použitelné na libovolně velkých skupinách studentů, GS2 jsou vzory aplikovatelné na skupiny do 40 osob. Také dělí vzory podle vztahu k jazyku. Lang1 označuje vzory nezávislé na jazyku v jakém se výuka vede, zda jde o mateřský či cizí jazyk. Několik málo vzorů, Lang2, je určeno pro výuku cizích jazyků.

Poslední forma kategorizace ([28]) rozlišuje podle momentu implementace vzoru (Moment of Implementation). Vzory implementované před začátkem celého kurzu, jsou označovány jako MoI1, vzory implementované před začátkem lekce jsou MoI2. Hlouběji rozlišuje vzory aplikované během lekce (MoI3) na subkategorie podle toho, zda jsou zaváděny na začátku (MoI3.1), v průběhu (MoI3.2) nebo na konci (MoI3.3) lekce. Vzory aplikované po konci lekci se řadí do MoI4, vzory použité po konci kurzu, v praxi pouze iterativní zlepšování kurzu, se řadí do MoI5.

3.5.1 Efektivní principy učení v hrách

Kapitola vychází z knihy *What video games have to teach us about learning and literacy* ([23]), kde autor rozebírá jádro úspěšných a účinných principů výuky, vycházejících z počítačových her, aby bylo možné se z nich inspirovat při školní výuce. Popsané principy jsou prokazatelně elementárními prvky efektivní výuky. Hry popisuje jako dobře navrženou zkušenost, ze které se lze učit a učitele označuje za návrháře těchto zkušeností.

Tvrdí, že nelze číst slova bez zkušenosti, čeho se týkají v reálném světě, a zároveň se nelze ze světa naučit nic relevantního k danému slovu, aniž bychom se o slovu již něco naučili. Základní hnací silou lidského učení je interakce s okolím, zařazováním slov do kontextů a schopnost mezi těmito kontexty přepínat. Efektivní učení je potom vhodnou kombinací vlastního zkoumání, analýzy prostředí a poskytnutých informací a získaných zkušeností.

Neřízené zkoumání se může ubírat libovolným směrem, proto je učení usměrňováním zkoumání. Úspěšné hry musí své hráče v první řadě naučit hru chápat a mít zájem o její dokončení. Proto stanovují jasný cíl hry, vybírají konkrétní cíle, které jsou dobře adaptovatelné hráči. Pozornost hráče poté navádí na jednotlivé prvky hry, tedy zkoumaného světa, aby nezabloudil a hru ztracen neopustil.

Popisuje častý problém výuky, poskytnutí abstraktní příručky, o kterou nikdo nežádá. Potom se chce po studentech plnit cvičení, aniž by znali cíl svého snažení. Pokud neznají cíl svého snažení, jen těžko mohou mít zájem na jeho plnění. Přitom existuje řada velmi komplexních her, které se dokáží naučit i studenti základních škol, přičemž obdobně složité předměty jsou odsunuty až do středoškolské výuky. Problém přijímání a chápání látky studentem evidentně nespočívá v její složitosti, ale v samotném jádru lidského učení. Student nemá s čím si informace spojit a jejich vstřebávání se mu jeví jako bezúčelné.

Hrám se daří replikovat prostředí pískoviště. V jádru jde o prostředí s nízkými riziky, kde je možné libovolně experimentovat bez obav z následků. Učíme se nejlépe zkušeností, v reálném světě může být zkušenost drahá, když někdo skočí ze skály a zabije se, tak se mu jeho zkušenost příliš nevyplatí. Uvědomování si těchto rizik vede k tomu, že se ve svém zkoumání často omezujeme a některá validní řešení nikdy neobjevíme.

Stručně popsané principy z knihy můžeme najít ve stávající výuce, jejich cílená aplikace by měla napomoci plynulosti výuky:

- **Aktivní kritické učení**. Prostředí, ve kterém se hráč pohybuje, funguje způsobem vyžadujícím aktivní myšlení, nikoli pasivní přijímání informací. Pouze pozorovatelé přejímají informace pasivně, účastníci se potřebují naučit v daném prostředí pohybovat a zjistit míru, jakou mohou ovládat. Pro výuku tuto roli zastává poskytnutý framework hry, nebo předpřipravená kostra hry.
- **Distribuce**. Význam, znalosti a návody k řešení jsou rozprostřeny po nástrojích, objektech, účastnících hry i po celém prostředí. Není žádný centrální zdroj vědění, větší poznání nabízí více znalostí.
- **Explicitní informace na vyžádání a informace právě včas**. Drobné instrukce, které mají hráči pomoci zvládnout danou situaci, jsou ve hrách předány právě v momentě výskytu dané situace. Pokud se hráči nedaří na danou informaci patřičně zareagovat, může se do stejné situace vracet znovu a znovu, dokud se nenaučí daný vstup patřičně zpracovat. Časem se naučí rozpoznávat situaci bez bezprostředního popudu a reagovat samostatně. Hry na rozdíl od škol na své účastníky také nesvalí hromadu informací naráz, pokud je nějaká rozsáhlá informace důležitá, je uložena stranou, v dostatečně přehledné a dostupné formě, aby v momentě její relevance se mohl hráč na daný zdroj obrátit a nastudovat si ho.
- **Identita**. Ve hře se hráč rozhoduje za virtuální postavu, vnitřně ale považuje postavu za svojí, jelikož rozhodnutí tvoří sám a vypovídá o vlastní osobnosti. Učí se také, jakým způsobem jeho rozhodnutí ovlivňují jeho postavu jako takovou.
- **Inkrementalismus**. Jednotlivé úrovně jsou uspořádány takovým způsobem, aby se hráč zpočátku učil generalizovat přínosné jednání, zatímco v pozdějších fázích hry musel přínosnost, správnost a použitelnost svých zažitých zkušeností zvažovat a přehodnocovat.
- **Inteligence materiálu**. Myšlení, řešení problémů a znalosti jsou uloženy v nástrojích, technologiích, materiálech a v prostředí. Při hledání problému proto není třeba se zatěžovat přemýšlením nad již nalezeným řešeními, stačí je aplikovat, proto má hráč daleko větší kapacitu na experimentování s řešením a na rekombinaci jednotlivých prvků k dosažení výsledku.
- **Intuitivní znalosti**. Intuitivní nebo tacitní znalosti bývají ve hrách naučeny opakováním. Takové naučené řešení je pak intuitivně použito v novém prostředí. Při tvorbě

nového prostředí nebo systému se lze inspirovat právě těmito naučenými prvky pro tvorbu snadno ovladatelné aplikace.

- **Koncentrace vzorku**. Vzorová situace se v rané fázi hry nebo při jejím zavedení do hry objevuje výrazně častěji než později v jejím průběhu. Účelem je, aby si hráč na daný prvek zvykl a naučil se ho identifikovat.
- **Kontextuální význam**. Jakékoli znaky, které hráč přijímá, nejsou obecně platné, nejsou samostatně jednoznačně interpretovatelné. Je nutné vyhodnocovat nejprve kontext v jakém se znak nachází, teprve následně lze určit jím přenášenou informaci.
- **Kontinuální učení**. Atraktivní úspěšné hry nejsou příliš repetitivní, právě naopak nutí hráče učit se stále novým věcem a upravovat svůj přístup k řešení problémů.
- **Kulturní model světa**. Hry využívají a reprezentují nějaký model světa, nemusí být nutně reálný. Hráč si v průběhu hry uvědomuje rozdíly jemu známého světa oproti světu ve hře. Jde zde typicky o odlišnosti osobností, možností a sociálních vztahů.
- **Kulturní model učení**. Řada her postupně odemyká jednotlivé vlastnosti a schopnosti skrze takzvaný výukový strom, hierarchii určující návaznosti schopností a požadavků k jejich osvojení. Hráči si uvědomují podobnost mezi učením v reálném světě a ve virtuálním světě.
- **Kulturní model sémiotiky**. Hráči si uvědomují, jaké znalosti vedou k osvojení významů jednotlivých termínů a značek, že se jedná o záležitost zkušeností a zájmů.
- **Mezitextové vztahy**. Samotné texty v různých kontextech mohou mít zcela odlišnou intepretaci, hráč se učí psané nebo mluvené instrukce zařazovat do jednotlivých kontextů. Je nutné poznamenat, že hra může mít více kontextů, zpráva může souviset s herní mechanikou, s navigací v prostoru nebo s ovládáním hry, pokaždé by stejná zpráva vyústila v jiné chování.
- **Multimodálnost**. Význam a znalosti nejsou spojeny s jedním typem znaků, ale jsou propojeny s řadou slov, zvuků, obrazů či dalších vjemů.
- **Návrh**. Používání již navržených systémů vede uživatele volně k jejich hodnocení, hráč je pak schopen identifikovat užitečné a kvalitní koncepty pro použití ve vlastním návrhu.
- **Odhodlané učení**. Hráči vynakládají velké úsilí na trénování dovedností vlastní virtuální identity, tedy postavy, za kterou hrají ve hře. Působením na virtuální svět skrze virtuální postavu, se vnitřně ztotožňují s jejich herní postavou a budují si vůči ní jisté závazky.
- **Podmnožina**. I počáteční výukové prostředí je podmnožinou nějakého reálného prostředí, s každou úrovní se tato podmnožina blíží více své reálné předloze.
- **Přemýšlení o sémiotických doménách**. Setkávání se se stejnými značkami různých významů nutí hráče přemýšlet o jejich vhodnosti, o jejich vzniku. Zpočátku ho může

použitá terminologie mást, ale účastí ve hře si na danou sémiotickou doménu zvykne a osvojí si ji.

- **Přenositelnost**. Znalosti a zkušenosti nabyté v jedné fázi hry jsou aplikovatelné i v dalších fázích. Občas s jistou úpravou či obměnou.
- **Psychologické moratorium**. V prostředí, kde jsou následky nižší než v reálném světě, je možné dovolit si rizikovější rozhodování, než jaké by bylo rozumné právě v reálném světě. V praxi si tak hráč může vyzkoušet libovolné řešení problému pouze za cenu ztráty času, nikoli jiných zdrojů.
- **Režim kompetence**. Hráč má vždy schopnosti potřebné na splnění daného úkolu. Aby se zlepšil a postoupil do další úrovně, měl by daný úkol splnit s maximálním vypětím svých sil. Může být omezen časem, počtem pokusů nebo jiným systémem. Bude dokola zkoušet současnou úroveň, než se bude moct přesunout do další.
- **Roztříštění**. Znalosti a vědění jsou sdíleny skrze komunitu dané hry, chybějící informaci nejspíše nalezneme mezi uživateli, nikoli mimo. Paralelu do běžného života si hráči vytvoří již sami.
- **Sebepoznání**. Využíváním poskytnutých nástrojů a postupem ve hře se hráč dozvídá o svých vlastních schopnostech a potenciálu. Řada studentů základních a středních škol si může myslet, že jejich kapacita na zapamatování si informací je velmi slabá, a tudíž je pro ně nemožné si zapamatovat veškeré evropské války posledního tisíciletí, ve hře ale po čase mohou zjistit, že si úspěšně pamatují daleko více lokací, předmětů a postav, než by považovali za možné.
- **Sémiotika**. Slova, zvuky, značky, obrazy nosí význam. Hráč se učí tento význam převzít, pochopit a eventuálně dovede význam podobným způsobem předat dále nebo upravit.
- **Sémiotické domény**. Různé značky mohou mít více situačních významů, ve hře se naučí hráč rozpoznávat různé domény a správně tyto značky vyhodnocovat. Postupně obsáhne danou doménu a podstatu, ze které vychází.
- **Sondování**. Učení je cyklem zkoumání vstupů a výstupů světa jako systému a tvořením hypotéz, jejich potvrzováním a vyvracením. Některé hypotézy je v průběhu hry nutné přehodnotit a následně upravit.
- **Stavba dovedností**. Jednotlivé dovednosti se učí interakcí s hrou, jejich opakováním a tříbením. Dovednosti jsou obvykle specifické pro různé herní žánry.
- **Text**. Pochopení textu přichází teprve v momentě, kdy je s ním spojen dostatek praktických poznatků ze zkoumané reality.
- **Trénink**. Ve hře se stále trénují patřičné dovednosti, aniž by byly pro účastníky nudné. Herní prostředí svým způsobem drží atraktivitu činnosti na vysoké úrovni po velmi dlouhou dobu.

- **Úspěchy**. Řada her implementuje kolekci úspěchů, které znázorňují hráči i ostatním, čeho ve hře již dosáhli a poskytují motivaci jak pro pokračování, tak uspokojení z již dosažených úspěchů, aby se proces, který k současnému stavu vedl, nezdál zbytečný.
- **Více řešení**. Cíle her jsou obvykle jasně dané, co ale dané být nemusí, je cesta. Hráč má na výběr, jakou strategii zvolí, jaké nástroje použije k dosažení svého cíle.
- **Zájmové skupiny**. Hráč objeví zájmové skupiny, tedy skupiny lidí, které svedly dohromady společné zájmy a cíle, nikoli rasové, etnické, pohlavní či věkové znaky.
- **Zasvěcení**. Hráč není pouze spotřebitel, zákazník, ale získává znalosti všech zúčastněných vývojářů, návrhářů, programátorů, grafiků, testerů a jiných a své zkušenosti může do jisté míry profilovat a vzdělávat se v problematice i v jiných směrech pouhou účastí.
- **Zesílení vstupů**. Učí se ve hře, že pouhé stisknutí tlačítka nás může posunout dále než běžný krok, bez velké obtíže lze přesunout velké množství předmětů. V reálném světě jsou tyto úkoly manuálně těžší, ale nic nám nebrání si práci usnadnit jinými nástroji.
- **Zkoumání**. Hry se nesnaží sdělit veškeré informace, vytváří tím prostor pro vlastní objevy, podporují přirozenou lidskou zvědavost.

3.6 Monolith2Projects

Nástroj Monolith2Projects je v této práci využit při zpracování doprovodného programu. Hlavním zaměřením této kapitoly je vysvětlení notací a konvencí obohajující zdrojový kód a jejich vliv na generování dílčích projektů.

Monolith2Projects je nástroj pro generování verzí projektu napříč jeho vývojem. R. Pecinovský jej používá ve svých učebnicích programování v Javě ([40], [42]) pro poskytnutí sady cvičení a projektů svým studentům.

Tento nástroj řeší problém správy zdrojových souborů pro postupně vylepšované programy, která není dobře podporována běžnými nástroji správy zdrojových kódů. Hlavním problémem bývá, že změny v úvodních projektech se buď mohou projevit v návazných projektech a tím často exponenciálně roste náročnost jejich údržby, nebo se rozhodneme změnu do dalších projektů nepropisovat, potom ale učíme na základě nekonzistentního příkladu.

Monolith2Projects tuto volbu nechává na autorovi dílčích projektů. Je zcela na něm, zda bude balíček, třídu nebo metodu dále využívat v návazných lekcích, nebo zda se k ní nehodlá vrátit. Pro identifikaci částí kódu, které se mají používat v různých lekcích, používá různé komentářové preprocesorové příkazy. Tím způsobem je možné celý postupně vylep-

šovaný projekt spravovat v jednom projektu naráz a jednoduše z něj generovat jednotlivá cvičení.

Praktická část této diplomové práce používá zejména dva základní typy komentářových preprocesorových příkazů. Oba příkazy jsou párové, první přidává obalený kód do vybraných projektů:

Výpis 1: *Definice příkazu pro přidání řádků do vybraných projektů*

```
1     String s;             // Tento řádek je zahrnut do všech projektů
2 //%A+ «5
3     s = "Kód vkládaný do projektů s ID<5, přeskočený v ostatních\n";
4 //%A-
```

Druhý přidává a odkomentovává obalený kód do vybraných projektů.

Výpis 2: *Definice příkazu pro odkomentování řádků ve vybraných projektech*

```
1     String s;
2 //%A%U+ «8
3 //  s = "Kód odkomentovaný pro projekty s ID<8, přeskočený v ostatních\n";
4 //%A%U-
```

Příkazy jsou uvedeny dvěma lomítky, tedy stávají se komentářem, a následovanými typem příkazu. Otevírací příkaz je označen znaménkem plus, případně i podmínkou pro omezení vybraných projektů a uzavírací příkaz je označen pomlčkou (mínus). Příkazy je možné do sebe vnořovat, je ale potřeba uvádět v jejich aktuální hierarchii.

Výpis 3: *Definice vnořených příkazů*

```
1     String s;
2 //%A+ «5
3     s = "Kód vkládaný do projektů s ID<5, přeskočený v ostatních\n";
4 //%A%A+ =2
5     s = "Kód vložený do projektu s ID=2, přeskočený v ostatních\n";
6 //%A%A-
7     s = "Pokračování kódu vkládaného do projektů s ID<5\n";
8 //%A-
```

Další definované příkazy podporují označení ignorovaných bloků prologu, epilogu nebo vybraných sekcí kódu. Důležitým příkazem je ukončení definic balíčku `//%P-`, který musí být povinně v každé třídě, za definicí balíčku tříd. Důvodem je, že generátor předcházející kód přeskakuje a doplňuje definici novou, na základě definice projektu v .m2p souboru.

Celý generátor včetně zdrojových souborů bývá běžně distribuován formou .jar balíčku. Aplikace po spuštění nabídne volbu jednotlivých projektů k vygenerování. Je proto potřeba nadefinovat, které soubory se mají zpracovávat do kterého z projektů.

K tomu účelu slouží soubor .m2p. Definuje jednotlivé projekty: jejich ID, názvy, zda a které projekty se mají zahrnout, z jakých balíčků se mají které třídy zpracovat, případně je možné jednotlivé artefakty zde přejmenovat nebo přesunout jinam ve struktuře výsledného projektu. Příklad zadání projektu:

Výpis 4: *Definice projektu a souborů do něj zahrnutých*

```
1  ▌=============================================================================
2  PROJECT ▌ 2_MapCreation ▌ Creating first classes, forming of a map
3  ▌-----------------------------------------------------------------------------
4  INCLUDE ▌ CM0_CanvasManager
5
6  PACKAGE ▌ cz.sedlacek.dp
7      FOLDER ▌ dp ▌ cz/sedlacek
8        Architect.java              ▌▌
9        BrickWallBuilder.java       ▌▌
10       FloorBuilder.java           ▌▌
11       ITileBuilder.java           ▌▌
12       StoneWallBuilder.java       ▌▌
13       Tile.java                   ▌▌
14       L1_Introduction.java        ▌▌
15       L2_MapCreation.java         ▌▌
```

Hlavička `PROJECT` odděluje jednotlivé definice projektů. Znak ▌ je zde používán k oddělování parametrů příkazů. Prvním parametrem projektu je kombinace jeho ID a názvu oddělených podtržítkem. Druhým parametrem je popis projektu.

Příkaz `INCLUDE` označuje dříve definované projekty, které mají být zahrnuty do tohoto projektu. Jeho parametrem je plný název požadovaného projektu.

Příkaz `PACKAGE` definuje, ze kterého balíčku budeme přebírat soubory. Prvním parametr určuje zdrojovou cestu, třetí cílovou cestu. Druhý parametr je volitelný a není zde uveden. Návazný příkaz `FOLDER` označuje složku, ze které budeme čerpat. Může být využit jak pro konverzi zdrojových souborů, tak pro přímé kopírování jiných pomocných souborů, například obrázků. Jeho první parametr uvádí cestu ke zdrojové složce. Následuje seznam souborů ze zdrojové složky, které chceme konvertovat do daného projektu.

Projekt Monolith2Projects je třeba sloučit s vyvíjeným projektem a vytvořit z nich spustitelný .jar. Pro správné fungování může být potřeba úprava konfiguračního souboru projektu Monolith2Projects.

3.7 Analýza námětů

Tato sekce se věnuje námětům příkladů používaných v kurzech programování. Náměty jsou navzájem kombinovatelné, ale propojení většího počtu námětů nelze doporučit, výsledný příklad by mohl být příliš složitý pro vstřebání, nebo i distraktující. Uváděné silné a slabé stránky, příležitosti a rizika se vztahují k námětům jako takovým. Je nutné upozornit, že se budou měnit s konkretizací daného námětu. Doporučeným přístupem před zvolením vlastního tématu je nastavit si pravidelné revizní body a provést detailnější zpracování zejména rizik a příležitostí nad vymyšleným konceptem.

3.7.1 Adventura

Velmi častým námětem programů pro výuku programování je adventura. Může se jednat o jednoduché i komplexní programy. Student se může zapojit do tvorby základů, nebo pouze do vybraných aspektů při procházení hrou. Tento koncept výukové hry se používá i pro výuku problematiky nesouvisející s programováním. Adventury lze jednoduše rozšiřovat pomocí prvků gamifikace.

Námět je dobře použitelný jak pro výuku algoritmizace, tak pro výuku návrhových vzorů, je třeba ale věnovat pozornost jakým směrem se cvičení ubírá, je zde tendence stále opakovat stejné přístupy. Výsledná aplikace by mohla být sice široká obsahově, ale rozsahově velmi omezená.

Oblíbené variace tohoto námětu jsou textové RPG (Role Playing Game), konverzace mezi uživatelem a hrou probíhá na základě klíčových slov vyměňovaných skrze konzoli, tahové RPG (nebo také Point-Click Adventure), které používá sérii statických graficky zpracovaných místností. Komerčně nejúspěšnější jsou akční RPG, kombinující prvky adventury a zcela akčního prostředí.

Silné stránky:

- Relativně nenáročné na přípravu základu programu
- Flexibilita komplexnosti programu
- Dobrá škálovatelnost rozsahu
- Lze vytvořit atraktivní statickou i dynamickou hru

Slabé stránky:

- Námět je náchylný na příliš detailní zpracování a tím i větší nároky na pracnost
- Směrů, kterými se studenti mohou vydat, je mnoho, často je nutné jejich úsilí stále hlídat a směrovat

Příležitosti:

- Další rozšiřování programu může pomoc znázornit dobrý nebo naopak špatný konceptuální návrh

Rizika:

- Rozhodnutí studentů, ohledně cílových funkcí programu, provedená na začátku vývoje mohou vést k exponenciálně rostoucí komplexitě programu, může být nutné některé funkce v průběhu redukovat, aby byla práce v rámci výuky zvládnutelná

3.7.2 Akční hra

Akční hry jsou mezi výukovými materiály obvykle zastoupeny ve spojitosti s vývojovým herním prostředím třetích stran, jako je například Alice, GameMaker nebo Prog&Play. Důvodem bývá náročnost tvorby podobného frameworku, který by byl dostatečně přehledný a použitelný studenty v semestrálním kurzu. Tento způsob ale není příliš dobře přijímán v případě vysokoškolských kurzů, které se snaží studenty naučit používat robustnější vývojová prostředí. Pozitivním aspektem akčních her je, že se skládají zejména z funkčních aspektů, což podporuje výuku návrhových vzorů.

Atraktivita akčních her závisí na řadě proměnných. Vizuální aspekt programu hraje velkou roli. 3D aplikace jsou na pohled atraktivnější, ale 2D jsou výrazně jednodušší na vývoj. Proto se u her používá takzvané 2,5D, režim, kdy hra je efektivně dvourozměrná, ale různé grafické modely a efekty vytváří dojem trojrozměrnosti. Aspekt ovládání hry je zásadní zejména pokud je třeba aby student hru samostatně zkoušel a opravoval v ní chyby. Pokud by ovládání hry nebylo příliš responzivní nebo intuitivní, mohlo by její používání vést ke zbytečné frustraci studenta.

Poslední zásadní proměnné, které je třeba se věnovat, je množství herních prvků. Hry obsahující značné množství herních prvků jsou často nešťastně navržené a uživatelé mají tendenci značnou část hry ignorovat. Dobře navržené hry s velkým množstvím těchto herních prvků, bývá těžké se naučit. Je nutné si uvědomit, že cílem výuky není učit se hru, proto je vhodné hru minimalizovat a zjednodušit na minimální atraktivní koncept. Jeho rozšíření může být náplní práce studenta ve volném čase.

Různá zpracování námětu akčních her se odvíjejí v jedné rovině od formy 2D nebo 3D, v druhé rovině od převládající herní mechaniky. 2D hry jsou v zásadě skákačky (neboli platformery, s pohledem z boku) a hry s pohledem shora (top-down games). 3D hry rozlišují mezi First-person (pohledem očima postavy) a Third-person (pohled zpoza hlavní postavy). Převládající herní mechanikou, zejména u 3D her, je střílení (tedy Shooter hry), stále se ale drží v oblibě i hry, kde je hlavní mechanikou pohyb (například skákačky).

Silné stránky:

- Návrhové vzory mají obvykle silnou logickou vazbu mezi reprezentací a účelem, jejich použití bývá intuitivní
- Programy postavené na komerčních nástrojích pro tvorbu her bývají velmi atraktivní pro studenty
- Studenti si při vývoji hry postupně zkouší nově přidané funkce a mohou postup dobře pozorovat

Slabé stránky:

- Náročné na přípravu frameworku nebo základního programu
- Atraktivní návrh obvykle využívá složitějších grafických elementů

Příležitosti:

- Námět je ideální pro předměty zaměřené na návrh a tvorbu her

Rizika:

- Jednoduché verze akčních her mohou na studenty působit primitivně, neohrabaně a nezajímavě
- Pokud ovládání hry v raných fázích není připraveno a je třeba aplikaci testovat pouze pomocí skriptů, tak mohou studenti ztrácet motivaci svou hru testovat detailně a zanechat v ní řadu defektů, které se nakumulují a projeví později. Následkem může být silná demotivace

3.7.3 Kvíz

Málokdy se využívají pro výuku programování, spíše mají za cíl ověřit znalosti studentů netradičním a atraktivním způsobem. Integrují se do výuky jako gamifikační prvek, nenahrazují běžné testy.

Při návrhu kvízů je vhodné si uvědomovat, jaké jsou pozitivní a negativní akce z pohledu uživatele. Člověk se přirozeně brání ztrátám a hledá příležitosti k zisku. Pokud účast ve hře představuje riziko ztráty (například snížení hodnocení z kurzu), uživatelé budou spíše aktivitu vynechávat, přestože nabízí odměny. Řada komerčních her proto implementuje elementy, které umožňují uživateli získávat předměty, přestože nemají žádnou hodnotu ani pro samotnou hru ani pro uživatele a zároveň se vyhýbají situacím, kdy by uživateli cokoli odebírali. Proto by tyto kvízy v ideálním případě měly pouze nabízet body navíc ve výuce a neměly by být součástí povinných aktivit pro hodnocení výuky.

Silné stránky:

- Jednoduchý a logický návrh
- Studenti se mohou inspirovat běžnými testy na které jsou zvyklí
- Velmi se blíží klasickým aplikacím

Slabé stránky:

- Námět nemusí být pro studenty atraktivní
- Komplexita programu je silně omezena jeho účelem

Příležitosti:

- Program může být v kurzu prakticky vyzkoušen studenty při průběžném znalostním testu
- Program může být použit při zahájení kurzu jako ukázka a inspirace nových studentů

Rizika:

- Program může být nutné silně modifikovat, z pohledu jeho použití, jinak může být potřeba představit jiné náměty pro podporu výuky

3.7.4 Logické, puzzle

Do tohoto námětu také spadá populární koncept únikových her (escape room), virtuálních či reálných. Jejich cílem v naprosté většině bývá naučit nebo procvičit schopnosti studenta v oblasti řešení problémů. Občas si tyto hry kladou za cíl učit algoritmizaci a vzácně nabádají studenty k tvorbě těchto her. Právě tvorba logických a puzzle her by byla vhodným přístupem k výuce metodikou Architecture-first.

Logické hry se často objevují jako součást adventur. V několika případech jsou tyto prvky použity k diverzifikaci obsahu cvičení ([31]) podle osobnostních typů. Tento námět byl úspěšně použit například v online kurzech CodeCombat pro výuku kódování a základní algoritmizace v postupných lekcích rozšiřováním základní knihovny funkcí.

Silné stránky:

- Téma je pro většinu studentů atraktivní ze své podstaty
- Vyřešení úlohy pozitivně motivuje studenty pokračovat a pouštět se do složitějších úkolů
- Námět lze kombinovat s jinými náměty

Slabé stránky:

- Námět může být velmi náročný na přípravu

Příležitosti:

- Koncept logických překážek a zkoušek nemusí být nutně součástí samotného programování. Lze ho aplikovat na fázi návrhu řešení studentem (obvykle vhodnější pro techničtější kurzy)

Rizika:

- Seřadit logické úlohy od nejlehčí po nejtěžší není jednoduché. Ještě těžší je zajistit postupnou návaznost růstu náročnosti. Pokud se nepodaří podpořit učící křivku studentů, skoky v náročnosti nahoru či dolů mohou studenty od úlohy silně demotivovat
- Stimulace náročnosti úkolu může být modifikována striktním zadáním, jako je například maximální počet kroků nebo použitých proměnných. Taková zadání se ale jeví jako odtažená od praxe a mohou zájem studentů o jejich řešení dále snížit

3.7.5 Ovladač, správa

Také označované jako manažerské hry. Koncový uživatel se obvykle snaží vybírat a přidělovat úkoly ke zpracování v závislosti na podmínkách a událostech v herním prostředí. Tento koncept je velmi zajímavý, ale málo zastoupený ve výuce programování, přestože nabízí téměř výhradně prostor pro implementaci jednoduchého až komplexního rozhodování, závislostí a postupů. Jelikož tento námět není prozkoumán příliš do hloubky, nelze jednoznačně určit, zda je dobře aplikovatelný na postupně rozšiřovaný program. Přístup by mohl být alternativou kurzu CodeCombat pro postupnou výuku návrhových vzorů.

Variace tohoto námětu zahrnují řadu her na efektivní organizaci času nebo akcí. Jiné se zaměřují na zpracovávání více úloh současně (multitasking). Některé hry mají za cíl nalézt a upravit optimální trasu pro přepravu ve stále se měnícím prostředí.

Silné stránky:

- Lze aplikovat na hry i na běžné aplikace (tvorba administrativy systému)
- Velmi se blíží aplikaci návrhových vzorů v praxi

Slabé stránky:

- Řada užitečných návrhových vzorů nemusí být dobře a logicky aplikovatelných v tomto námětu. V takovém případě by bylo vhodné výuku rozšířit o jiný, odtržený příklad pro výuku vybraných konceptů

Příležitosti:

- Tento námět může rozšiřovat již existující aplikace, pokud by byly navrženy s ohledem na implementaci ovladače nebo správce

Rizika:

- Dobré řešení problému může podporovat spíše repetitivní úkony. Rozsah aplikace by proto musel být velmi dobře hlídán a její vývoj směrován

3.7.6 Praktické příklady aplikací ze života

Řada kurzů programování používá příklady aplikací každodenního užití, kalkulačky, mediatéky nebo testeru. Nevýhodou přístupu praktických aplikací bývá, že část studentů konkrétní příklad nedovede nadchnout, ale to samé může platit i u námětu hry. Na praktických aplikacích se sice dobře demonstrují konkrétní návrhové vzory, ale často je těžké jich více napojit rozumně na stejný příklad, a proto je potřeba připravovat více příkladů pro plynulou výuku.

Studenti ochotně používají jim relevantní praktické případy, málokdy se ale skupina studentů shodne na relevanci jednoho příkladu. Pro větší úspěch je možné tuto variantu nabídnout a individuálně korigovat.

Silné stránky:

- Aplikaci lze velmi dobře upravit pro demonstraci konceptů a návrhových vzorů
- Někteří studenti ocení praktický přínos zaměření kurzu

Slabé stránky:

- Pro účely kurzu může být potřeba kombinovat více aplikací pro pokrytí látky kurzu, nebo značná modifikace základní aplikace
- Praktické příklady nemusí být motivující pro řadu studentů

Příležitosti:

- Studenti mohou navrhovat témata aplikací, které by jim přišly osobně zajímavé nebo využitelné. Taková možnost ale klade vysoké nároky na přípravu a vedení kurzu

Rizika:

- Při použití univerzálního námětu, shodného pro celou skupinu studentů, může značná část studentů cílit pouze na minimální možné zadání, bez jakékoli další vlastní snahy o zlepšení nebo rozšíření obzorů

3.7.7 Simulace

V simulacích se obvykle vymodeluje prostředí a jeho chování, není nutné, aby tuto část vytvářeli studenti. Na studentech je, aby vytvořili pravidla, skripty, které upraví chování jistých elementů a dostanou simulaci do cílového stavu. Může se jednat o simulace dělení buněk, hledání únikových cest, zvěře hledající potravu a podobné. Volba kontextu a cíle simulace určuje, zda se výuka bude zabývat spíše algoritmizací nebo aplikací vzorů. Prvky simulace lze používat při rozšiřování mnoha typů aplikací pro tvorbu automatizovaného chování nebo umělé inteligence.

Silné stránky:

- Lze navrhnout příklady na míru cílům výuky
- Námět lze minimalizovat a nechat studenty soustředit se pouze na vybrané oblasti nebo problémy

Slabé stránky:

- S komplexností příkladu rostou nároky na jeho přípravu

Příležitosti:

- Simulace lze zakomponovat do strategických her a her v reálném čase
- Studenti si mohou snadno vyzkoušet efektivnost jejich algoritmů a hledat efektivnější přístupy

Rizika:

- Výpočetní náročnost může růst exponenciálně, výsledná vizualizace nebo výpočet může být časově náročný a zpomalovat práci studentů

3.7.8 Strategie

Strategické hry jsou obvykle vnímány jako simulace bitev. Hráč ovládá jednotky v boji, staví obranu nebo ji prolamuje. Používají pohled shora a ovládací prvky myši a klávesových zkratek. Pro většinu strategií je omezující síla procesoru, a to zejména v závislosti na počtu instancí jednotek ve hře. Proto se využívá separátní logika pokrývající pohyb objektů a sledování polohy, která je studentům poskytnuta a do které se dále nezasahuje.

Existuje řada variací strategických her. Pro výuku nejlépe adaptovatelné jsou RPG strategie, kde uživatel ovládá pouze jednu postavu, podobně jako v adventurách. Alternativou je věžová obrana (tower defense), zaměřená na obranu pomocí několika typů obranných věží.

Klasické strategie se soustředí na bitvy, války a správu základny, běžně si hráč nejprve staví základnu, následně armádu a vlastním tempem se snaží naplnit cíle mise. Z klasických strategií byl odvozen formát 4X strategií, kde 4X znamená explore, expand, exploit, exterminate neboli prozkoumat, rozvinout se, využít, vyhubit. Jedná se zejména o kolonizační hry, které z principu nelze dobře balancovat pro hru více hráčů. Veškeré uvedené variace mohou mít provedení jak tahové strategie, tak v reálném čase (RTS, real time strategy). Tahové strategie jsou jednodušší na vývoj, ale hledání chyb a jejich oprava může být velmi únavná.

Silné stránky:

- Nabízí širokou škálu adaptací
- Umožňuje značný podíl vlastní kreativity ovlivnit základní elementy hry, aniž by se změnil její celkový návrh

Slabé stránky:

- Velmi vysoká náročnost na přípravu prostředí nebo frameworku

Příležitosti:

- Prostředí připravené pro tvorbu strategických her, je možné znovu použít jako základ pro tvorbu řady her založených na dříve zmíněných námětech, například: adventury, akční hry, různé 2D hry

Rizika:

- Námět ztrácí popularitu mezi komerčními hrami, je proto možné, že nebude příliš populární mezi studenty

4 Výběr námětu

Při výběru námětu se v první řadě věnuji slabým stránkám a rizikům pro prvotní eliminaci málo vhodných námětů. Přetrvávající náměty dále řadím dle vhodnosti, a to na základě jejich silných stránek. Protože praktická část této práce má za cíl koncept demonstrovat, tak pro samotný výběr námětu použitelného pro tuto práci nejsou příležitosti námětu natolik zásadní, aby rozhodnutí nutně ovlivňovali. Lze k nim určitě přihlédnout a mělo by se k nim přihlížet, pokud se již vytváří cvičení pro použití ve výuce.

4.1 Posouzení námětů dle slabých stránek a rizik

Slabinou adventur je potenciál jejich komplexnosti a náchylnost na zbytečné komplikace v případech kdy studentům dáváme volnou ruku při tvorbě. Dopad na návrh ale tato slabina nemá. Přenáší se pouze spolu s rizikem nadměrné nutnosti hlídání do dalších fází využití, kde může ovlivnit úspěšnost celého projektu zavedení nového příkladu pro výuku. Adventury jsou tedy pro účely této práce plně validním námětem.

Akční hry mají mezi slabými stránkami náročnost na přípravu frameworku nebo základního programu. Potřeba takového základu aplikace zvedá náročnost praktické ukázky, ale práce do ní vložená není dále ani vidět ani není využitelná. Podobně tomu je tak s atraktivností návrhu, věnovat se mu zde nepřináší další hodnotu a je neefektivní. Námět akčních her je tedy z tohoto výběru vyřazen.

Kvízy bývají obecně jednoduchými aplikacemi. Zda bude či nebude atraktivní, není natolik důležitým aspektem pro samotný návrh. Ale je zde riziko, že nebude možné dobře a logicky navázat výuku některých prvků nebo návrhových vzorů na tento námět. Řešením takové situace by bylo výrazně ohnout užití a podobu aplikace nebo cvičení doplnit o izolovanou ukázku. Takovému případu by bylo dobré se vyhnout, proto je i tento námět z výběru vyřazen.

Logické hry a puzzle bývají těžší na přípravu, co se týká přípravy jednotlivých řešitelných úloh. Náročnost klesá, pokud je úloha pouze jedna nebo sami studenti danou úlohu vytváří. Návrh příkladů této práce proto příliš neomezuje. Riziko s neúměrně rostoucí náročností příkladů se při tomto návrhu přenáší na implementaci samotných cvičení. Logické hry a puzzle jsou tedy dalším použitelným námětem pro návrh praktických cvičení.

Tvorba ovladačů a aplikací na správu dat je sice velmi prakticky založená. Ale tato práce si klade za cíl zejména podporovat výuku návrhových vzorů, jejichž začlenění do takové aplikace může být náročnější než u jiných uvedených námětů. Proto nebude námět ovladače dále zvažován pro výběr.

Potíží aplikací s praktickým využitím bývá, že často využíváme pouze malé množství programovacích konstruktů nebo návrhových vzorů, abychom dosáhli svého cíle. Ačkoli práce na samotné aplikaci může být pro řadu studentů atraktivní, může být nutné daný příklad opustit za účelem představení jiných konstruktů nebo vzorů. Již pouze z tohoto důvodu je lepší se tomuto příkladu vyhnout, pokud cílíme na výuku rozšiřováním jednoho základu aplikace. Samozřejmě pokud bychom se rozhodli aplikaci z praxe ve finále použít, bylo by dobré věnovat se tomu riziku, že pro studenty nemusí být vybraný námět zcela atraktivní. Zejména se jedná o situace, kde je sofistikovaná verze dané aplikace velmi rozšířená, nebo pokud již existuje řada frameworků pro tvorbu podobných aplikací, což mohou být například e-shopy.

Simulace, ať už se jedná o simulace reálného světa nebo fiktivního bývaly velmi atraktivní, což lze odvodit z faktu, že vznikly předměty na ně přímo zaměřené. Ale popularita těchto předmětů ukazuje, že toto téma s nejvyšší pravděpodobností nebude vhodné pro vstupní kurzy a širokou škálu studentů. V praxi stále můžeme vidět úspěšné simulace použité i v základních kurzech, tyto simulace bývají obvykle velmi jednoduché a pokrývají látku několika praktických cvičení, nikoli celého semestru. Právě z důvodu, že se zde cílí na celosemestrální výuku a spjatý příklad by mohl jednak neúměrně růst v náročnosti a klesat na atraktivitě, by bylo vhodné se tomuto námětu vyhnout. Nelze ale říct, že by byl nepoužitelný, dozajista je možné se jím inspirovat a zakomponovat jeho prvky do finálního námětu.

Strategie tak jak je zná většina studentů, jsou časově náročné hry, vyžadující obvykle detailní znalost rovnováhy hry. Pokud hra není vyvážená, nebo trvá příliš dlouho, ztrácí na atraktivitě. Tento trend lze pozorovat na trhu počítačových her, kde tento žánr téměř vymírá. Přidáme-li k této informaci fakt, že strategické hry jsou obecně náročné na přípravu základu aplikace, tak je nejlepším rozhodnutím tento námět vyřadit z výběru.

Na základě slabých stránek a rizik můžeme vyřadit značnou část námětů, pro účely této práce si nyní vybíráme zejména mezi adventurou, logickými a puzzle hrami a simulacemi. Z nichž adventura je velmi často používaný námět a v podmínkách VŠE v Praze je to i námět pro semestrální práci vstupního kurzu. Pouze z tohoto důvodu nebude adventura pro příklad použita, ale pro zbytek výběru námětu bude stále uváděna.

4.2 Posouzení silných stránek námětů

4.2.1 Adventury

Adventury jsou v porovnání s jinými náměty velmi jednoduché na přípravu a zároveň atraktivní pro studenty. Nejjednodušší formy adventur jsou textové nebo takzvané point-

click adventury. Textové adventury jsou ovládány pomocí textových řetězců, často ani neimplementují vizualizaci hry, pouze vypisují současný stav. Point-click adventury vykreslují scénu a hlídají akce kliknutí myší v definovaných oblastech a na jejich základě volají patřičné metody, spouštějí požadované akce.

Pro adventuru není složité přidávat nové funkce k současnému návrhu. Koncept pohybu nebo změny prostředí zvýší komplexitu dané aplikace, ale zachová její hlavní myšlenku. Příklady lze libovolně upravovat podle současných potřeb, například se rozhodneme implementovat nějakou formu inventáře, můžeme ho považovat za bezedný pytel věcí, nebo můžeme přidávat podmínky k jeho využití. Lze omezit jeho kapacitu jedním či více atributy. Tedy omezit počtem věcí, jejich váhou nebo velikostí. Předměty můžeme také kategorizovat a stanovit, že jediné, co lze do batohu přidat je jídlo.

Rozsah adventury je také zcela na našich potřebách. Hra se může odehrávat v jedné místnosti nebo v několika. Můžeme pracovat se záchytnými body, s více úrovněmi, nebo vše demonstrovat pouze v jedné. Hráči můžeme poskytnout několik základních schopností, typu jít na místo, rozhlédnout se, sebrat nebo zahodit předmět. Můžeme také přidat interakci s jednotlivými předměty, která může být reprezentována jednoduchým úkonem „prozkoumat“ nebo několika typy akcí pro různé skupiny předmětů od přepínání spínačů po nasednutí na koně.

Zejména u mladších studentů, statické prostředí nebývá příliš atraktivní. U adventury je možné přidat grafické rozhraní a ovládání klávesnicí a myší namísto sady příkazů. Z tahové hry lze relativně jednoduše vytvořit hru v reálném čase, alespoň pokud jsme ve fázi návrhu a stále můžeme upravit základ příkladu těmto potřebám.

4.2.2 Logické hry a puzzle

Již samotný námět říká, že se jedná o výzvu, něco, co je možné vymyslet obvykle bez hlubokých znalostí nějaké problematiky, ale pouze následováním několika jednoduchých pravidel. Přístup k řešení puzzle her je stejný jako k řešení zadaných úloh. Stejné metody a vzorce chování budou studenty provázet celým jejich minimálně studiem, často i celým životem. I pokud jejich úkolem zde je vyhodnocovat správnost cizího nebo obecného řešení, stále se vystavují riziku vytvoření chyby v řešení a nutnosti prozkoumat jeho správnost.

Obecná koncepce logických úloh je, že řešíme nejprve jednoduché úlohy a přecházíme na složitější, klademe si další překážky a omezení, abychom poznali své limity. Zadostiučinění z vyřešené úlohy se odvíjí od její složitosti. Pokud je řešení problému samotnou podstatou práce, nikoli pouze jejím příznakem, tvoří pak značnou část motivace. V tomto bodě je vhodné se dívat i na současnou situaci a dlouhodobý dopad filosofie zdroje motivace. V praxi většina zaměstnanců sleduje, kolik práce vložili do projektu a kolik uživatelů se s jejich výtvorem setká. Hodnotí sami sebe metrikami jako je míra používání jejich díla,

přestože jejich podíl na dané službě nebo produktu je minimální. Právě tito zaměstnanci mají problém se vyrovnat s tím, že projekt selže, je zastaven a nikdo jejich dílo neuvidí. Obvykle si stěžují že jejich práce k ničemu nebyla. Což by byla pravda pouze v případě, že do projektu sami investovali. Reálně odvedli požadovanou práci a byli za ni zaplaceni. Nespokojenost by zde mohla být oprávněná z pohledu investora nebo zadavatele, který právě ztratil velkou část své investice, nebo projektového řízení, které na svém resumé má nyní neúspěšný projekt a s tím může i částečně nést vinu za jeho selhání. Z pohledu zaměstnance se žádná hodnota neztrácí, naopak si z projektu odnáší zkušenosti. A právě tyto logické úlohy a puzzle by mohli pomoci studentům adoptovat mentalitu sebevzdělání, získání zkušeností, aby nehleděli na zadané úlohy jako na ztrátu času, pokud je nikdy nikdo znova neuvidí.

Logické úlohy a obecně výzvy je možné začlenit do mnoha jiných námětů a v ideálním případě i do samotného způsobu výuky. Můžeme studentům zadat jako úkol, aby vymysleli, jakým způsobem řešit zadanou úlohu. Je zde jisté riziko vyšší časové náročnosti při použití takového přístupu oproti výuce se striktním zadáním, kde se student nemůže příliš odchýlit od daného úkolu a nepotřebuje se hlouběji zamýšlet nad způsobem řešení. Máme minimálně dva další přístupy, jak jednoduše začlenit výzvy a logické úlohy do výuky. Jedním z nich, je navrhnout a připravit program k vylepšení jako sérii samotných úloh. Úkolem studenta může být doplnit do programu požadovanou funkcionalitu a následně i programové řešení dané úlohy neboli jednoduchou umělou inteligenci, která obecný problém vyřeší. Alternativně může být úkolem studenta vytvořit nebo upravit program tak, aby byl schopen vyhodnotit správnost řešení úlohy. V tomto případě nebývají úlohy pro studenty příliš jednoduché, může u nich trvat déle, než je nalezena chyba a je vhodné snížit četnost jednotlivých úloh oproti přístupu, kde student úlohy zejména řeší a kontrolu za něj obstará program.

4.2.3 Simulace

Krásnou vlastností simulací je, že si můžeme vybrat libovolnou situaci z reálného života nebo si vymyslet kompletně vlastní. Může se jednat o dělení buněk, tok uživatelů na webu nebo o šíření požárů. Přidáváme pravidla nebo požadavky podle vlastních potřeb a můžeme je zacílit přímo na podstatu konceptu nebo návrhového vzoru, který chceme představit. Pod simulací si můžeme představit nejjednodušší formu kalkulace, kde je naším výstupem sada středních hodnot nebo součtů veličin. Řešení také můžeme vizualizovat, ať už v daném momentu nebo v reálném čase. Pokud bychom měli zájem náš simulátor dále vylepšit, můžeme připravit uživatelům nástroje na ovlivňování simulace v jejím průběhu v reálném čase.

Pokud máme dobře připravené prostředí nebo základ aplikace, je možné do jisté míry nechat studentům volnou ruku nad výběrem předmětu simulace. Jejich simulaci poté redukovat, aby bylo použitelná pro účely výuky. Zde je nutné předem rozmyslet, co by měla

výsledná simulace splňovat. Demonstruji na zjednodušeném příkladu, cílem malého příkladu je naučit nebo procvičit návrhové vzory *Přepravka*, *Stav* a *Jedináček*. Pro vzor *Přepravka* můžeme požadovat, aby se prvky simulace slučovaly do párů, ale mohly dále jednat nezávisle jeden na druhém. Pro vzor *Stav* potřebujeme, aby se jednotliví účastníci simulace měnili, co se týče chování nebo vlastností, mezi minimálně třemi stavy. Pro vzor *Jedináček* potřebujeme, aby simulace obsahovala jednoho účastníka, který bude unikátní, v ideálním případě pro demonstraci vzoru, by tento účastník neměl být permanentně přítomný a jeho účast by měla být opakovaně vyžadována řadou událostí. Takové zadání by mohla splňovat simulace množení motýlů v uzavřeném prostředí. Na motýlovi samotném vyzkoušíme vzor *Stav*. Dokud je housenkou, můžeme sledovat stavy nakrmený a nenakrmený. Následně můžeme implementovat jednotlivé fáze životního cyklu jako stavy. Pro udržení životního cyklu simulovaných živočichů, můžeme definovat vlastní sociální chování pro motýly, řekněme že tvoří páry a jako páry se účastní migrace nebo si dělí práci, ale o potravu si obstarají samostatně, nikoli v párech. Případně můžeme definovat kmenové rozdělení a vzor *Přepravka* využít právě pro rozlišení jejich příslušnosti. *Jedináčkem* v tomto příkladu může být predátor, který populaci sledovaných živočichů reguluje.

4.3 Vlastní návrh námětu

V první řadě se zde vybírá námět na základě již popsaného posouzení silných stránek, následně se popisuje jeho vlastní forma. Forma je rozvedena v kapitole Navrhovaný postup výuky, kde rozvíjí námět logické hry, hledání průchodu bludištěm.

Zavrhuje se zde námět adventury. Ty sice jsou v mnoha ohledech flexibilní, ale proto jsou také velmi často používány při výuce. Řada dobrých typů adventur a jejich implementací pro různé kurzy, s různými cíli výuky je snadno dohledatelná a nebylo by příliš přínosné vymýšlet další adaptaci stále stejného námětu.

Simulace jsou na druhou stranu používané spíše zřídka, obvykle až ve specializovaných kurzech. Jakým způsobem lze k jejich návrhu přistupovat popisuje posouzení jejich silných stránek. Můžeme si takto navrhnout simulaci na míru výukové lekci nebo celému předmětu a řadu jejích variací. Vzhledem k tomu, že se simulacemi často zabývá úzce zaměřený předmět, ponechám tento námět specifikem daného předmětu a dám přednost logickým a puzzle hrám.

Příklad je navrhován pro vstupní kurzy, proto by měl začínat jednoduchými úkoly. Zpočátku se tedy zaměří na základ celého konceptu. Student je v roli tvůrce logické hry, uživatel programu by měl řešit úkol. Tím úkolem bude navigovat bludištěm z bodu A do bodu B. Pro jednoduchost se doporučuje držet čtvercové mřížky, jak je celkem běžné, představíme zde vzor *Přepravka* pro popis pozice na plátně. Vytvořit čtvercové bludiště, vyznačit počátek a konec. V tento moment ani není potřeba vytvářet nějaké překážky, protože se snažíme zprovoznit hlavní funkci a to vyhodnocení, zda byla nalezena cesta k cíli.

Základním stavebním prvkem je posun kurzoru podél svislé nebo vodorovné osy vždy o jedno pole. Tím se zajití, že vytváříme nepřerušenou linii od bodu A. V momentě dosažení bodu B byla nalezena cesta neboli řešení předloženého problému. Toto nalezené řešení ale nemusí být nutně správné. Cesta by měla vést okolo překážek, nikoli skrz. Procházení překážkami vyřešíme tak, že využijeme *Prostředníka* k ovládání pohybu, který ohlídá, aby se uživatel nedostal na stejnou pozici jako je překážka. Nyní jsme schopni vytvořit bludiště a říct, že pokud se uživatel dostal do cíle, tak nalezl správné řešení. Ukázka námětu je znázorněna na obrázku Ukázka labyrintu (zdroj: hra The Witness – 2016) níže.

Základ je připraven, nyní lze přidávat komplikace do řešení. Jednou z prvních komplikací jsou povinná pole. Označíme předem několik polí, skrz které musí cesta vést a při dosažení cíle zkontrolujeme jejich stav. Aby tato komplikace skutečně ztěžovala řešení, je nutné aplikovat pravidlo řešení jedním tahem. Jedním tahem je zde myšleno, že se uživatel nemůže vrátit na pozice kde již byl.

***Obrázek 1**: Ukázka labyrintu (zdroj: hra The Witness – 2016)*

5 Navrhovaný postup výuky

Kapitola rozvádí vybraný námět, hledání cesty labyrintem, do detailního postupu průchodu lekcemi. Výklad pojmů a konceptů je zcela ponechán na pedagogovi, práce se mu nijak nevěnuje. Pro demonstraci konceptu je k práci přiložen generátor projektů (viz kapitola 3.6), který generuje jednotlivé stavy postupně vylepšovaného programu.

Generátor používáme k přípravě projektu pro studenty, zpravidla se věnujeme cvičením označeným pouze číslem. Ta znázorňují stav projektu na konci lekce. Pokud je potřeba zahájit lekci s upraveným projektem, v porovnání s konečným stavem z lekce minulé, tak je v generátoru také uveden projekt XB, kde X reprezentuje číslo projektu, který je upraven.

Při výuce lze procházet seznamem cvičení s tím, že pokud je následující cvičení označeno pouze číslem, je na studentech, aby projekt dovedli do stavu následujícího cvičení. Pokud je následující cvičení označeno číslem a písmenem B, nahradíme studentům stávající projekt projektem následujícím. Přechody mezi používanými projekty jsou v jednotlivých lekcích komentovány.

Jednotlivé lekce mají definovány cíle a zavedené pojmy, shrnuté v tabulce 1: Tabulka 1: Navrhovaný postup výuky. Zda a jakým způsobem budou studentům pojmy vysvětleny, ponechává práce na vyučujícím. Naplnění cílů lekce dosahujeme podle sekce popis lekce. Je nutné si uvědomit, že lekce nejsou navrženy na základě konkrétního prostředí výuky (středoškolské, vysokoškolské či celodenní školení) a je tedy na implementátorovi výuky, aby bral ohled na délku jednotlivých cvičení a lekce do nich spojoval či rozděloval.

Případné podněty pro úpravu výuky na základě kontextu výuky, schopností studentů a komentáře k některým rozhodnutím učiněným v dané lekci jsou uváděny na konci lekce, ke které se vážou, v sekci poznámek.

Tabulka 1: Navrhovaný postup výuky

Lekce	**Cíle lekce**	**Zavedené pojmy**
Lekce 1 – Úvod	Seznámit studenty s vývojovým prostředím, zde s BlueJ Seznámit studenty se základními koncepty objektově orientovaného programování Prakticky vyzkoušet práci s projektem v daném prostředí	Object Class Method Variable Instance Reference Variable types Return Crate
Lekce 2 – Tvorba mapy	Seznámit studenty s pojmy veřejnými a privátními atributy, konstruktory a konceptem přetěžování metod Vyzkoušet návrhový vzor *Stavitel* Využít příležitosti pro představení rozhraní a dědičnosti Představit pomocnou třídu a návratový typ metody void	Public, Private Constructor Overloading Builder Interface Override Void Utility class
Lekce 3 – Tvorba cesty hráče	Provést studenty řešením problému v programu Představit a vyzkoušet návrhový vzor *Jedináček* Znovu použít koncept Stavitele	Static, Final This Singleton
Lekce 4 – Příprava podmínek výhry	Představit studentům základ vzoru *Pozorovatel* Procvičit znalost vzoru *Jedináček* Připravit podklad pro zavedení seznamů, jejich procházení a podmínek	Observer If
Lekce 5 – Tvorba pravidel	Seznámit studenty se seznamy Dokončit implementaci *Pozorovatele* Seznámit studenty s hodnotou null Představit vzor *Prázdný objekt*	List For Null NullObject
Lekce 6 – Úprava vyhodnocení konce hry	Představit studentům vnitřní třídy Vysvětlit a vyzkoušet vzor *Stav*	State pattern Inner class

5.1 Lekce 1 – Úvod

Úvodní lekce do programování je velmi dobře zpracována již v knize *OOP – learn object oriented thinking and programming* [42], proto využijeme této knihy do její kapitoly 9: *The Expedition into the Interior of Instances* a navážeme na ni kapitolou Lekce 2 – Tvorba mapy. Změna této úvodní kapitoly by neměla přidanou hodnotu pro samotnou výuku. Jedná se o představení základů, terminologie a orientace ve vývojovém prostředí.

Cíle úvodní lekce tedy shrnují potřebný základ potřebný pro navazující kapitolu. Jelikož je postup výuky ve zmiňované knize velmi podrobná, tak je v této kapitole uveden i popis úvodní lekce, který poskytuje náhled na stěžejní části úvodní lekce.

Cíle lekce

- Seznámit studenty s vývojovým prostředím, zde s BlueJ
- Seznámit studenty se základními koncepty objektově orientovaného programování
  - Pojmy objekt, třída, metoda, reference, proměnná, typy proměnných
- Prakticky vyzkoušet práci s projektem v daném prostředí
- Rozšířit znalosti studentů o návratové hodnoty, výčtové typy, typy proměnných, návrhový vzor *Přepravka* a jejich praktické vyzkoušení

5.1.1 Popis lekce

Návrh výuky pracuje s vývojovým prostředím BlueJ, je tedy nutné, aby studenti již měli prostředí nainstalované, případně si ho na začátku výuky nainstalovali. Poté se mohou začít seznamovat s nutnými základy k používání daného prostředí.

Pro popis nástroje, je vhodné začít načtením nějakého projektu, obrazovka se tím naplní daty, nad kterými lze vysvětlovat jednotlivé aspekty aplikace. Z těch jsou prozatím podstatné: diagram tříd, průzkumník objektů a možnost kompilace.

V této fázi návrh používá přímo Canvas Manager, který ve své výuce používá R. Pecinovský, je samozřejmě možné ho zastoupit jeho variantou. Základní koncept, který rozvíjíme je znázorněn níže v diagramu.

Studenti si získané teoretické znalosti o prostředí a jednotlivých pojmech (třídy, objekty, metody) mohou ihned vyzkoušet. Výhodou prostředí BlueJ je možnost ihned vidět vytvořené objekty a dynamicky je přiřazovat. Výhodou zmíněného Canvas Manageru je paralelní vizualizace objektů v dalším okně.

V první řadě studenti vytvoří několik objektů a vyzkouší si volat připravené metody. Pro vytváření objektů jsou vhodné ty zobrazované na plátně, jako jsou obdélníky, trojúhelníky nebo elipsy. Vhodné metody jsou potom například přesuň, změň velikost nebo barvu.

Aby se studenti nesnažili objekty v první řadě zejména parametrizovat numerickými hodnotami, je vhodné představit třídy reprezentující jejich vlastnosti, jako například `NamedColor` nebo `Position`. Tyto dvě třídy také slouží k ukázce využití výčtového typu a vzoru *Přepravka*.

Pokud je možné již v tento moment odhadnout schopnosti studentů, můžeme je vzít v potaz při dané lekci. Méně zkušené studenty je dobré nejprve seznámit s koncepty zachycení barvy a pozice pomocí již připravených tříd a později je nechat si vytvořit vlastní výčtový typ a *Přepravku*. Zkušenější studenty se můžeme pokusit seznámit s konceptem a nechat je rovnou doplnit těla metod v daných třídách.

Během práce s pozicemi, případně barvami nebo velikostmi, seznamujeme studenty s referencemi na objekty a proměnnými. Při této příležitosti je možné vyzdvihnout důležitost a typy proměnných, zejména ve spojitosti s návratovými hodnotami. K procvičení je vhodné například nechat studenty posouvat objekty pomocí metody `moveBy(dx: int, dy int)` v momentě kdy se již pozice zachycuje pomocí třídy `Position`. Student si zde musí uvědomit, že musí nejprve z instance pozice dostat obě číselné proměnné a poté je správně přiřadit do metody.

5.2 Lekce 2 – Tvorba mapy

Zde začínáme s tvorbou samotné logické hry, ve které hráč hledá cestu z jednoho bodu v bludišti do druhého. Tato lekce, tvorba mapy, pouze připravuje podklad a samotnou strukturu bludiště.

Lekce pracuje s projektem 1_Introduction, který lze vygenerovat z přílohy práce, generátoru projektů z monolitu. Je vhodné zvážit varianty přístupu k výuce této lekce. Varianta použitá v praktické části neposkytuje studentům připravené třídy, ale nechává na lektorovi, aby studenty jejich vytvořením provedl. Důvodem je, že si studenti aktivitu lépe zapamatují, pokud si jí vlastnoručně vyzkouší.

Alternativním přístupem by bylo poskytnout prázdná těla tříd pro studenty, aby do nich pouze doplnili vlastní kód. Prázdná těla tříd studentům poskytuje projekt 1B_MapCreation_empty. Důvod proč tento přístup není v praktické části používán, je že budí u studentů dojem drobné úpravy programu, nikoli tvorby nové aplikace. Zejména pragmaticky založení studenti bez předchozích zkušeností s programováním vyžadují, aby byl vidět výsledek jejich práce a zprovoznění nedodělané třídy je obvykle neuspokojí.

Cíle lekce

- Seznámit studenty s pojmy veřejnými a privátními atributy
- Seznámit studenty s konstruktory a konceptem přetěžování metod

- Vyzkoušet návrhový vzor *Stavitel*
  - Využít příležitosti pro představení rozhraní a dědičnosti
  - Pojmy implements a override
- Představit pomocnou třídu a návratový typ metody void

5.2.1 Popis Lekce

V lekci tvoříme mapu bludiště. Začneme jednoduše, stačí vytvořit mřížku tří svislých uliček na tři vodorovné. Ty chceme obklopit rámečkem. Vytváříme tedy čtverec 7x7 polí.

Požadavky na jednotlivá pole jsou, abychom byli od sebe schopni vizuálně odlišit volná pole a stěny. Zároveň bychom chtěli tuto vlastnost někde uchovávat, aby ta informace byla jednoduše přístupná. Proto vytváříme další *Přepravku*, kterou zde nazýváme `Tile`.

Pro třídu `Tile` je zásadní, aby nám zprostředkovávala informace o jejím obsahu, tedy objekt, který představuje a zda se jedná o volné pole či nikoli. Z objektu samotného jsme potom schopni získat informace o jeho poloze, tvaru, barvě či velikosti, dle potřeb.

Právě sem můžeme zasadit lekci o privátních a veřejných atributech. Studenti tedy vytváří metody na získání a nastavení jednotlivých proměnných.

Zavedením polí nám roste náročnost na tvorbu mapy dlaždic. Pokaždé vytváříme nový `Rectangle`, zadáváme parametry a přiřazujeme objekt do `Tile`, kde zadáváme další parametr a na závěr voláme metodu `show()` nad původním tvarem. Proto si práci usnadníme *Stavitelem*.

Staviteli přednastavíme jednotlivé atributy, které by výsledná dlaždice měla mít a pomocí metody `createWall()` dlaždici vytvoříme. Mohli bychom připravit univerzálního stavitele, kterému by se nastavily výchozí hodnoty a on by vytvářel dlaždice daného typu, dokud bychom výchozí hodnoty neupravili. Chtěli bychom ale využít příležitosti a sjednotit více tříd pod jedním rozhraním, aby je mohla využívat pomocná třída `Architect`.

Proto pro každý typ dlaždice vytvoříme separátního stavitele. Metoda `createWall()` bude používat parametr `Position`, aby nebylo nutné pole dále upravovat. Nyní můžeme opět využít příležitosti a zabývat se přetěžováním metod. Místo pozice lze zadávat souřadnice polí, ať už podle bodů (0, 50, 100, 150…) nebo podle sloupců (0, 1, 2, 3…). Také lze připravit metodu pro zadání odlišné barvy, velikosti nebo tvaru.

Jednotlivé stavitele sjednotíme pod rozhraním `ITileBuilder`. Cílem je získat jednotně pojmenovanou metodu pro tvorbu dlaždic včetně seznamu jejích variant a standardních proměnných. Na atributu velikosti dlaždice lze vyvolat nutnost dodatečné definice metod definovaných rozhraním. V příkladu je použita výchozí hodnota velikosti dlaždice reprezentující podlahu. Pokud je přidána dlaždice podlahy jako první, veškeré další objekty jsou

přidány přes tuto základní dlaždici. Proto může zabírat celé okno a není třeba velikost upravovat ani doplňovat metodu pro zohlednění potřeby jiné velikosti.

Abychom nemuseli tuto mapu pokaždé sestavovat manuálně, využijeme třídu `Architect`. Jejím účelem je postavit vždy stejné základy mapy dlaždic, které můžeme dále rozšířit. Zde má `Architect` pouze roli automatizace tvorby dlaždic, není potřeba, aby dovedl zprostředkovávat jiné funkce, než je příprava jednotlivých částí mapy.

Jak vyplývá z class diagramu pro tuto lekci, `Architect` zde skládá mapu ze tří částí. Jendou metodou vytváří zem, po které je možné se pohybovat. Druhou připravuje vnější stěny, které se skládají z velkých bloků, nikoli z jednotlivých dlaždic. Třetí metoda staví vnitřní stěny. Tyto metody mají návratovou hodnotu `void`, protože jejich výsledek máme odzkoušený a případné chyby nás v rámci cvičení příliš netrápí.

Tyto podpůrné metody na závěr shrneme pod finální `buildMap()`. Metoda bude nadále používána v přípravné fázi jednotkových a zejména funkčních testů.

Stručný postup průběhu lekce

1. Vytvořit třídu `Tile` podle vzoru *Přepravka*
2. Vytvořit *Stavitele* `StoneWallBuilder`, `BrickWallBuilder` a `FloorBuilder`
3. Přidat rozhraní `ITileBuilder`
4. Implementovat rozhraní vytvořenými staviteli
5. Vytvořit pomocnou řídící třídu `Architect`
6. Implementovat metody třídy `Architect`, aby bylo možné postavit základní labyrint jedním příkazem

5.2.2 Cílový stav

Diagram tříd 1: Tvorba mapy znázorňuje stav na konci této kapitoly. Třídy poskytnuté studentům, využívané bez prav, jsou znázorněny žlutě. Nově přidané třídy jsou znázorněny zeleně. Praktická část obsahuje projekt 2_MapCreation, který je reprezentací tohoto diagramu a stavu projektu po dokončení této kapitoly.

Podstatné je, aby byly vytvořeny a implementovány následující třídy:

- `Tile` – pro zachycení objektů na mapě
- Rozhraní `ITileBuilder` a alespoň dvě třídy implementující toto rozhraní
- Pomocná třída `Architect` – pro zjednodušení tvorby mapy

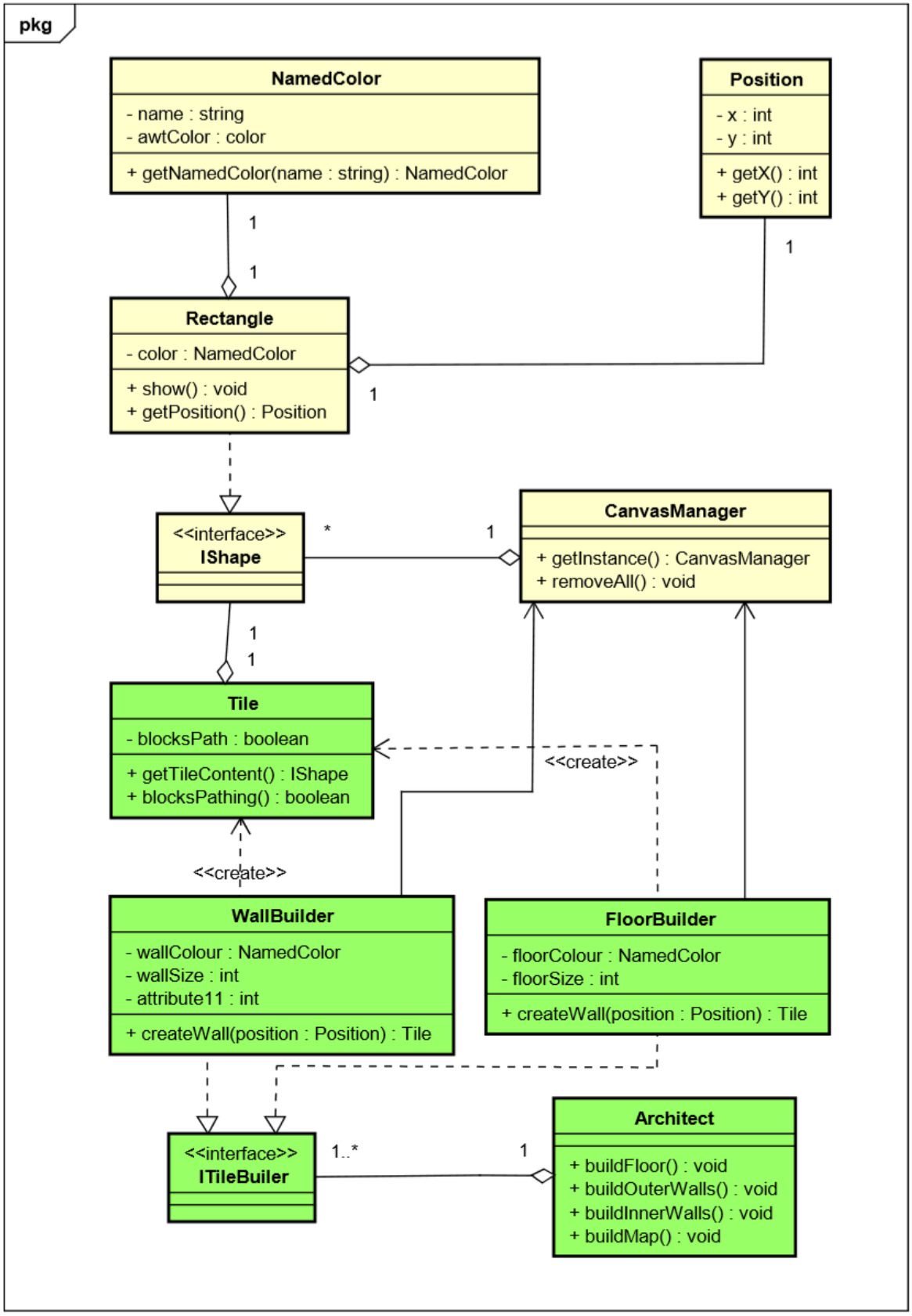

***Diagram tříd 1**: Tvorba mapy*

Testovací třída `L2_MapCreation` shrnuje jednotlivé milníky lekce. Prvním milníkem je vytváření mapy pomocí jednotlivých tvarů, což je výchozím stavem. Dalším milníkem je vytváření dlaždic za pomoci prvního stavitele.

Je nutné si uvědomit, že mezi prvními milníky může být v závislosti na vedení praktických cvičení značná časová mezera zejména z důvodu manuálního zkoušení tvorby mapy a přípravy jednotlivých tříd. Přece jen je zde navržen přístup, ve kterém studenti vytváří celé třídy, nikoli pouze doplnění metod. Pokud tedy nechceme příliš omezovat rychlost a směr postupu studentů, tak z časových důvodů může být nutné zde cvičení rozdělit a případně si definovat další drobné milníky nebo samostatné práce pro studenty.

Následně je zachycena tvorba mapy pomocí všech připravených stavitelů. Finální test tvoří mapu několika málo příkazy skrze již zmíněnou pomocnou třídu.

Poznámky

Pro ověřování správné funkčnosti je vhodné využívat jednotkové testy. Studenti je zde nebudou nutně využívat pro testování komponent ani jejich integrací v pravém slova smyslu, ale pro funkční systémové testování, stejným způsobem jako jsou použity třídy `L1_Introduction` a `L2_MapCreation`.

Pokud by studenti používali jednu třídu jednotkových testů pro zachycení více scénářů, bude nutné nastavit do sekce po ukončení testů doplnit volání metody `removeAll()` třídy `CanvasManager`, aby se na plátně nehromadily objekty z různých testů. Je možné, ale nedoporučoval bych, nechat studenty tento problém objevit samostatně. Jedná se o logický problém vázaný na používání třídy `CanvasManager`, její znalost nelze od nových studentů očekávat.

Rozdělení vnitřních a vnějších stěn v programu se zavádí z důvodu, že vnější stěny budou vždy neměnné, na rozdíl od vnitřních, které se můžeme rozhodnout přeskupit, odstranit nebo dynamicky přidávat.

Můžeme také již v tento moment definovat alternativní verze jednotlivých dlaždic, například světlé a tmavé verze pro dekorativní účely. Tuto deviaci je možné později použít pro kontrolu správného zacházení s vyhodnocováním ekvivalentních dlaždic.

5.3 Lekce 3 – Tvorba cesty hráče

Lekce pracuje s projektem 2B_CharacterPathCreation_inclGamePlan. Jeho provedení lze opět získat z generátoru projektů. Projekt je shodný s 2_MapCreation, s jediným rozdílem, a to je třída `GamePlan`, kterou v novém projektu studentům poskytneme. Třída je znázorněna v diagramu tříd červenou barvou. Do této třídy studenti nemusí zasahovat. Následně vytvo-

řené funkce programu jsou navrhovány s cílem, aby se studenti mohli k třídě `GamePlan` chovat jako k černé skříňce.

V předchozí kapitole jsme ničím nepohybovali. Pouze jsme skládali jednotlivé předměty na plátno. Nyní vytvoříme prvky hry, které se budou v průběhu hry měnit. Což je v tomto případě trasa vytvořená hráčem, která hledá řešení vytvořeného labyrintu. Praktická část této práce proto používá třídu `Mover`, pro animaci pohybu, aby byla změna studentům patrná.

Cíle lekce

- Provést studenty řešením problému v programu
  - Doplnění požadovaného rozhraní nebo třídy
- Představit a vyzkoušet návrhový vzor *Jedináček*
  - Pojmy this, static a final
- Znovu použít koncept *Stavitele*

5.3.1 Popis lekce

Lekci zahajujeme ve stavu, kde máme nově přidanou třídu `GamePlan`, která ovšem využívá zatím neexistující rozhraní `ITile`. Bylo by sice možné třídu upravit tak, aby využívala již zavedenou třídu `Tile` a veškeré herní objekty zavádět skrze tu samou třídu, ale bylo by to velmi nepřehledné.

Proto vytvoříme rozhraní `ITile`. Stále zde platí, že dlaždice mají obsah a mohou blokovat pohyb. Rozhraní implementuje existující třída `Tile` a případně další třídy se stejným účelem jako jsou `BrickWall` nebo `StoneWall`. Když jsme vytvořili rozhraní, můžeme přidat další třídu pro znázornění cíle.

Třída `Target` bude reprezentovat záchytný bod na mapě. Aby nebylo nutné ho při každém testu znovu nastavovat, využijeme třídu `Architect`, která za nás i toto pole pokaždé připraví. Je důležité odpovědnou metodu `setUpObjective()` volat v rámci přípravy jako poslední, jinak je možné, že bude cíl překryt jinou dlaždicí.

Nyní vytvoříme reprezentaci hráče. Podobně jako v úvodních lekcích, vytvoříme tvar a přidáme ho na plátno. Tvar můžeme přesouvat pomocí metod `moveRight()` a dalších, ale rychle si studenti všimnou, že připravený tvar může procházet všemi stěnami.

Tvorbě více instancí hráčů zamezíme tak, že vytvoříme hráče jako jedináčka, jak znázorňuje **výpis 5:** *Definice metody třídy Player*. Pro přípravu samotného objektu hráče můžeme pak využít metodu `initialize()`. Tuto metodu není vhodné volat z konstruktoru, bylo by

potom nutné se starat o zbývající reprezentace hráče při jeho opakovaném vytváření v rámci testů. Situaci později můžeme využít k představení vzoru *Stav*.

***Výpis 5:** Definice metody třídy Player*

```
1  public class Player implements ITile
2  {
3      private static final Player player = new Player();
4
5      private Player() {
6      }
7
8      public static Player getInstance() {
9          return player;
10     }
11 }
```

Aby bylo možné využít připravených dlaždic a atributu určujícího, zda blokují pohyb, je potřeba začít aktivně využívat třídu `GamePlan`. Veškeré vytvořené dlaždice je třeba do plánu přidat, lze tak učinit přímo v konstruktoru jednotlivých tříd, nebo při jejich vytváření. Protože je možné, že budeme chtít vytvářet dlaždice a nepřidávat je ihned do hry, tak využijeme stavitelů pro přidávání dlaždic i do herního plánu.

Nyní samozřejmě původní metoda pro posouvání tvarů bude stále posouvat hráče na obsazená pole. Proto využijeme metod třídy `GamePlan` pro posouvání tvarů. Můžeme si ověřit, že hráč nyní skrze stěny neprochází.

Studentům se nyní může v sériích postupných testů začít stávat, že by se některé dlaždice na mapě nezobrazovaly nebo by nebylo možné pohybovat s hráčem. Toto chování lze očekávat, jelikož `GamePlan` je jedináček a používáme stále stejnou instanci pro následné testy, zůstávají nám na mapě staré dlaždice. Proto stejně jako již v testech uklízíme objekty z `CanvasManager`, tak bychom měli vyprázdnit i objekty mapy metodou `removeAllTiles()`.

Stále pouze pohybujeme jedním tvarem a není patrné, kudy cesta hráče vedla. Rozhodujeme zde jakým způsobem bude logika hry nadále řešena. Pokud hráč dosáhl cíle, nemohl projít žádnou překážkou a můžeme tedy usoudit, že z počátečního bodu do cíle cesta vede a byla nalezena. Zachycení skutečné cesty by nemělo být potřeba.

My si ale v této hře chceme úkol ztížit. Hráč by měl dosáhnout cíle, aniž by prošel stejným bodem dvakrát. Stále nebude potřeba kontrolovat, zda jednotlivé body cesty na sebe navazují, jelikož herní plán nám neumožní dlaždice přeskakovat. Budeme již pouze kontrolovat, zda se hráč nesnaží dostat na místo kde již byl. Pro přehlednost na taková místa přidáme nový tvar, který bude zároveň překážkou.

Vytváříme tedy novou třídu `Trail` implementující rozhraní `ITile`. Pro snadnou vizualizaci je ideální použít stejný tvar a barvu, jako se používá pro hráče. Ideálním místem pro vytváření této cesty je třída `Player`, ta za sebou stopu zanechává. Rozšíříme tedy třídu o metodu zanechání stopy.

Je vhodné si v tento moment vyzkoušet, že se naše nová třída chová, jak očekáváme. Zůstává na místě, kde byl hráč, blokuje pohyb zpět a nevyvolává žádné kritické chyby. Pro účely testování může studentům pomoci stopu hráče barevně odlišit.

Následně zařadíme pokládání stopy do metody pohybu hráče. Vytváříme tedy v třídě `Player` metody pro pohyb do všech stran, kde hráč pokládá stopu a následně si přesouvá svůj tvar pomocí `GamePlan`.

Závěrem lekce ověříme funkční aspekt vytvořené cesty. Jsme tedy ve stavu kdy uživatel je schopen najít a znázornit cestu do cíle. Zároveň se nemůže vracet na již navštívená pole. Cíl může být zatím pouze dosažen, ale nic se samozřejmě neděje.

Stručný postup průběhu lekce

1. Zavést rozhraní `ITile`
2. Implementovat rozhraní pro třídu `Tile`
3. Přidat třídu `Target`
4. Upravit třídu `Architect`, aby vytvářela i cíl labyrintu
5. Vytvořit *Jedináčka* `Player`
6. Rozpohybovat třídu `Player` metodami pohybu skrz `GamePlan`
7. Přidat třídu `Trail` pro znázornění cesty uživatele
8. Přidat metody pohybu do třídy `Player` pro automatické přidání položky trasy

5.3.2 Cílový stav

Diagram tříd 2: Tvorba cesty hráče obsahuje žluté třídy z minulé lekce, červenou třídu přidanou na začátku této lekce, oranžové třídy upravované touto lekcí a zelené nově přidané třídy. Praktická část obsahuje cílový stav v projektu 3_CharacterPathCreation.

V této lekci je nutné upravit třídy `ITile` a `Architect` a dále přidat:

- Třídy `Trail` a `Target` implementující rozhraní `ITile`
- Třídu `Player` jako *Jedináčka*
- Začít využívat `GamePlan` pro přesouvání objektů na mapě

Testovací třída `L3_CharacterPathCreation` obsahuje jednotlivé funkčně integrační testy ověřující požadované chování v jednotlivých fázích lekce. Začíná s přípravou hráče a jeho pohybem po mapě. Následně ověřuje, zda můžeme využívat `GamePlan` k navádění hráče po mapě a zda mu je tím skutečně zabráněno procházet skrz překážky.

Finálně se soustředí na kontrolu hráčem vytvořené trasy. Je vhodné si ověřit, že je trasa skutečně zanechávána na herním plánu, objevuje se na správných místech a blokuje zpětný pohyb hráče. Dosažení cíle nevyvolává žádnou akci.

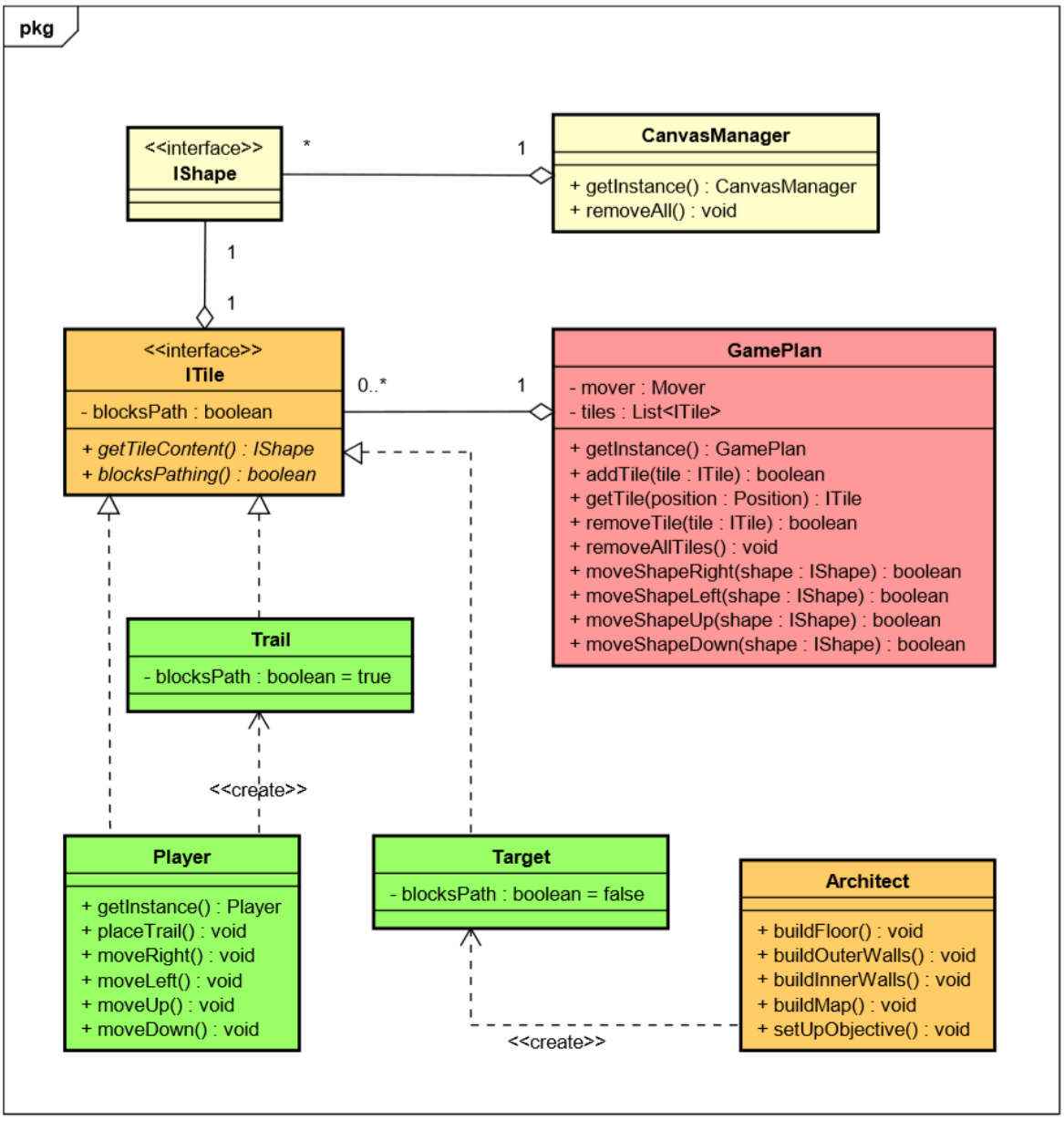

***Diagram tříd 2**: Tvorba cesty hráče*

Poznámky

V diagramu tříd jsou `ITile` a `Architect` vyznačeny oranžově. Tím se zde naznačuje, že je těmto třídám nutné věnovat pozornost, protože je budeme rozšiřovat.

Třída `ITile` nemusí mít jasné ukotvení do programu výuky. Mohli jsme ji přidat již v minulé lekci, mohli bychom ji přidat na začátku spolu s `GamePlan` a následně toto rozhraní implementovat stávající třídou `Tile` a novými třídami. Ale zde je navrženo, aby třídu doplnili sami studenti.

Hlavním důvodem je, že tím na začátku lekce, bude student postaven před upravenou verzi programu, která ale hlásí chybu. Toto rozhraní je používáno právě nově přidanou třídou `GamePlan`. Bude tedy nucen potřebám třídy vyhovět vytvořením rozhraní.

Třída `Architect` je zde rozšířena o metodu pro přípravu cíle labyrintu a používá k tomu jednu z nově vytvořených tříd, `Target`. Návrh v této práci patřičnou metodu doplňuje až po přípravě nové třídy.

Pokud by se zavedení `GamePlan` bez rozhraní `ITile` jevilo jako příliš matoucí pro studenty, lze počáteční konflikt vyvolat právě poskytnutím úpravy třídy `Architect`. Tento alternativní scénář ale praktická část této práce neukazuje, protože doplnění jednoho rozhraní by nemělo být příliš náročným.

5.4 Lekce 4 – Příprava podmínek výhry

Pracujeme s projektem 3_CharacterPathCreation. Uživatel vytváří cestu, ale po dosažení cíle se nic nestane. Vytvoříme podmínky a jejich vyhodnocování, aby hráč dostal potvrzení o splnění úkolu. Začneme s jednou podmínkou a přidáme další. Student by si měl před koncem lekce uvědomit, že by podmínky neměly být přiřazovány do hry napevno, ale zpracovávány dynamickou strukturou.

Testovací třída `L4_AddingObjective` poskytuje tři základní scénáře, kterými se budeme v této lekci zabývat. Prvním scénářem je dorazit do cíle, aniž bychom prošli záchytným bodem, ověřujeme jeho nutnost. Druhým scénářem je dorazit do cíle a následně do záchytného bodu. Logicky by měla hra po dosažení cíle končit výhrou nebo prohrou. Třetí je pozitivní scénář, ten prochází nejprve záchytným bodem a následně jde do cíle. Poslední scénář kontroluje, zda po neúspěšném pokusu je hra skutečně obnovena do počátečního stavu a lze ji dokončit.

Cíle lekce

- Představit studentům základ vzoru *Pozorovatel*
- Procvičit znalost vzoru *Jedináček*
- Připravit podklad pro zavedení seznamů, jejich procházení a podmínek

5.4.2 Popis lekce

Lekci začínáme tím, že definujeme podmínku splnění cíle hry. Cílem je dosáhnout záchytného bodu, vizualizovaného již v minulé lekci. Úvahou zde je, že pokud je tvar reprezentu-

jící cíl vidět, tak nebyl dosažen. Pokud je něčím překryt, znamená to, že bylo dosaženo cíle.

Vytváříme tedy třídu `CheckpointCondition`. Tato třída má mít pro nás zásadní metody určující, zda je podmínka splněna a znovu vyhodnocení dané podmínky. Metoda `isMet()` pouze vrací zaznamenaný stav. Metoda `update()` stav vyhodnocuje.

Vyhodnocení změny stavu připravené podmínky zatím umíme nejlépe vyhodnotit při každém pokusu o pohyb. Proto budeme metodu update volat z metod třídy `Player`. Podmínka není pouze jedna, plánujeme přidávat další, a proto se nemůžeme odkazovat na její statickou instanci. Třídu hráče rozšiřujeme o metodu nastavující pozorovatele a o metodu upozorňující na možné změny.

Připravené metody `setObserver(CheckpointCondition cond)` a `notifyObserver()` slouží zatím zejména ke kontrole správné funkčnosti. Při přípravě mapy po vytvoření cíle, přidáme cíl jako pozorovatele našeho hráče. Z metod pohybu bude podmínka průběžně vyhodnocována.

Abychom poznali, zda byla vyhodnocena, můžeme nastavit akci po splnění. Pokud nechceme zatím představovat kondicionály, můžeme využít třídy `IO` a její metody `endIf(boolean end)`, která ukončí aplikaci, pokud byla podmínka splněna.

Následně chceme přidat další záchytný bod. Tím nám roste počet pozorovatelů a není vhodné je nadále řešit z třídy `Player`, tu bychom rádi udrželi přehlednou. Připravíme proto třídu `ConditionManager`, jedináčka, který si bude naše záchytné body pamatovat a upozorní je na pohyb hráče.

`ConditionManager` přebírá roli pozorovatele z pohledu hráče, původní metoda `notifyObserver()` bude od tohoto bodu směrovat na správce podmínek. Již v tento moment bychom si měli zvolit cíl trasy a vytvořit pro něj separátní třídu `FinalTileCondition`, abychom ji mohli využít při následných kontrolách.

V první fázi zavádění správce podmínek můžeme kontrolovat obě podmínky zároveň. Pokud jsou splněny, vypíšeme si správu pomocí `IO.inform(String text)`. Nyní nezávisí na pořadí naplnění podmínek, můžeme tedy nejprve dojít do konce trasy, následně se vrátit do záchytného bodu. I v takovém případě jsme cíle dosáhli.

Následně upravíme logiku vyhodnocování podmínek. Zda byly kontrolní body dosaženy je zajímavou informací pouze v moment, kdy bylo dosaženo cíle. Při každé změně tedy kontrolujeme pouze podmínky ukončení a pokud ty jsou splněny, zkontrolujeme také zbývající podmínky. Pokud i tyto podmínky byly splněny, zobrazí se zpráva, že cíle hry bylo dosaženo.

Tím se ale stále neřeší problém, že můžeme z cíle pokračovat dále a podmínky postupně splnit. Z toho důvodu do kontroly ostatních podmínek přidáme alternativní příkaz, který hráči sdělí, že byl neúspěšný, pokud doje do cíle, aniž by splnil ostatní podmínky.

Posledním krokem lekce, je po neúspěšném pokusu hru obnovit do výchozího stavu a oznámit hráči, aby zkoušel řešení znovu. Student si zde musí uvědomit jaké kroky jsou nutné a jaké nadbytečné. Pokud odpovídá architektura hry návrhu, stačí odstranit z herního plánu veškeré dlaždice a nechat nového architekta znovu vytvořit mapu. Odstranit obrazce z plátna není zásadní (pouze doporučené), protože při tvorbě mapy budou překryty.

Stručný postup průběhu lekce

1. Vytvořit třídu `CheckpointCondition` pro vyhodnocení dosažení cíle
2. Aplikovat vzor *Pozorovatel* na vztah mezi pohybem hráče a vyhodnocováním podmínek
3. Rozšířit počet kontrolních bodů vytvářených třídou `Architect`
4. Vytvořit třídu `ConditionManager` pro správu kontroly podmínek
5. Rozlišit podmínku cílového pole od podmínek mezi bodů

5.4.3 Cílový stav

Diagram tříd 3: Příprava podmínek výhry žlutě znázorňuje neměnné třídy z minulých lekcí, oranžově upravované třídy z minulých lekcí a zeleně nově zaváděné třídy. Řešení lekce je zachyceno v praktické části projektem 4_AddingObjective.

Lekce rozšiřuje třídy `Player` a `Architect`, aby braly v potaz nutnou tvorbu a zpracování podmínek. Pro vyhodnocování podmínek je přidat:

- Třídu `ChecpointCondition`
- Správce podmínek `ConditionManager`
- Třídu `FinalTileCondition` pro nastavení konceptu jednoho cíle

Na konci této lekce by měl student dostat zpětnou vazbu programu, zda byly podmínky pro nalezení cesty splněny. Podmínkami se zde rozumí projetí záchytných bodů.

Poznámky

Lekci je možné zjednodušit vynecháním třídy `FinalTileCondition`. Je možné využívat striktně `CheckpointCondition` a pouze označit za finální vybrané pole v `ConditionManager`. Tuto změnu je ale dobré provádět zejména v moment, kdy je cílem zkrátit lekce a zjednodušit podmínky hry vynecháním kontroly cíle a jejím nahrazením kontrolou záchytných bodů.

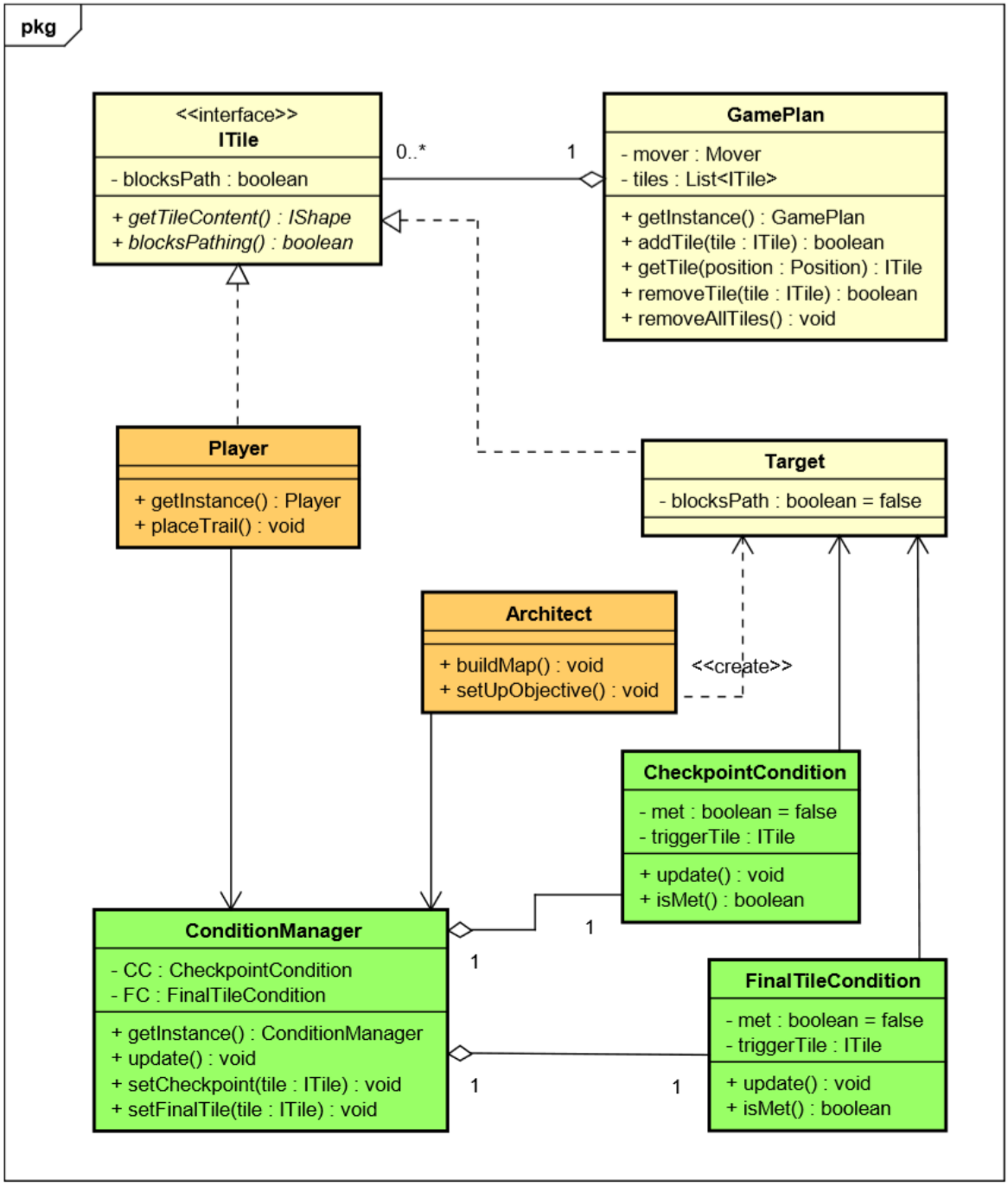

***Diagram tříd 3**: Příprava podmínek výhry*

Cílem námětu je vyhodnocovat více různých podmínek, pokud tedy vynecháváme kontrolu posledního dosaženého pole, měli bychom nahradit tuto podmínku nějakou jinou. Nejjednodušším příkladem je zakázané pole, tedy pole, skrze které hráč nesmí procházet.

Třída `FinalTileCondition` může být dále využita pro rozšíření hry. Je možné stanovit více možných finálních polí. Při dosažení jednoho z nich, by se vyhodnocovaly pouze ostatní podmínky, nikoli zbývající finální pole. V takovém případě by se logika vyhodnocení zásadně lišila od `CheckpointCondition`.

V této lekci navržený postup výuky zavádí metody `setObserver()` a `notifyObserver()` na zdánlivě málo strategickém místě. Důvodem je tento koncept, který plánujeme dále rozvíjet v následujících lekcích, studentům představit v jednoduché formě s jistým opodstatněním. Dokud se jednalo o třídy jedináčky, mohli jsme bez obav volat tyto jedináčky.

Nyní si nemůžeme být jisti danou podmínkou a v rámci delšího testování ani její permanencí. Dále si uvědomíme, že je pro nás jednodušší tuto logiku přesunout na třídy jako je `ConditionManager`. To pro zatím nemusí být zřejmým a přínosným řešením z pohledu studenta.

5.5 Lekce 5 – Tvorba pravidel

Pracujeme s projektem 4_AddingObjective. Vytváření proměnné pro každou podmínku je velmi nepraktické, proto se zaměříme na práci se seznamy, do kterých podmínky vložíme a po dosažení cíle zpracujeme. Nyní navíc program ztížíme požadavkem, aby v labyrintu bylo více cílů.

Abychom zbytečně nekontrolovali dlouho řadu podmínek výhry, budeme držet dva separátní seznamy na cíle labyrintu a na podmínky splnění úkolu. Pokud je dosaženo jednoho z cílů, zkontrolují se ostatní podmínky. Zároveň se v tento moment rozhodne, zda byla úloha vyřešena, nebo ne. Nepovolíme hráči pokračovat dále v labyrintu, pokud dosáhl jednoho z cílů.

Cíle lekce

- Seznámit studenty se seznamy
- Dokončit implementaci pozorovatele
- Seznámit studenty s hodnotou null
- Představit vzor *Prázdný objekt*

5.5.1 Popis lekce

Studentům vysvětlíme, že bychom chtěli veškeré podmínky držet v seznamech, aby bylo možné labyrint rozšířit a přidávat do něj další podmínky, bez nutnosti úprav třídy `ConditionManager`. Proto představujeme seznamy a koncept práce s nimi.

Aby bylo možné do seznamů vkládat univerzální záznamy podmínek, vytvoříme rozhraní `ICondition`. To implementují třídy podmínek z minulé lekce, `FinalTileCondition` a `CheckpointCondition`.

Začneme s transformací třídy `ConditionManager`. Založíme dvě nové proměnné `List<ICondition>`, jednu pro koncové body, druhou pro podmínky. Následně připravíme potřebné metody pro jejich správu, tedy možnost přidávání a odebírání podmínek a vymazání celého seznamu.

Upravíme metodu kontroly cílových bodů, `CheckFinalConditions()`, tak aby prošla seznam koncových bodů a uložila do proměnné podmínku, která je splněná. Nemusí nás zajímat, zda nebylo splněno více podmínek, jelikož kontrola ostatních podmínek probíhá v moment dosažení prvního cíle a v ten moment se rozhoduje o výsledku hry. Situace, že by hráč dosáhl dalšího z konců tedy nastat nemůže.

Následně upravíme metodu kontroly dalších podmínek, `CheckOtherConditions()`. Tento seznam je pro účely hry dobrovolný a nemusí obsahovat žádné speciální podmínky. Proto se ve výchozím stavu chová, jako by byly veškeré podmínky splněny. Proto nastavíme v metodě výchozí stav na splněno a tvrzení budeme vyvracet průchodem seznamu podmínek. Pokud narazíme na nesplněnou podmínku, zachováme tento fakt do proměnné a po dokončení cyklu, se bude na základě této proměnné rozhodovat, zda uživateli oznámit výhru nebo restartovat hru.

Odstraníme pozůstatky stavu z minulé lekce, takže třída `ConditionManager` již pracuje pouze se seznamy. Tím způsobíme konflikt ve třídě `Architect` týkající se metody `setUpObjective()`. Původní metody pro registraci podmínek byly odstraněny, je potřeba připravit nové.

Postup nastavení podmínek hry nyní zahrnuje vytvoření tvaru pro reprezentaci cíle, vložení tvaru do dlaždice, registrace dlaždice do herního plánu a nově vytvoření podmínky a přidání podmínky do správce podmínek.

Je vhodné zde ověřit, že program stále funguje, stejně jako dříve. Po této zkoušce pozměníme složení labyrintu. Odstraníme fyzické překážky vytvářené ve třídě `Architect` metodou `buildBrickWalls()` a přidáme do bludiště další koncový bod.

Je důležité nechat studenty ověřit funkčnost všech koncových bodů, bez dalších podmínek, pro případ, že seznam cílů vyhodnocují chybně, nebo vůbec ne. Když koncové body fungují korektně, můžeme do mapy vložit více podmínek typu kontrolní bod. Nyní ověříme i správnou kontrolu průběžných bodů cesty, zda jsou skutečně všechny body povinné a zda jsou korektně vyhodnoceny.

Nyní víme, že se program chová správně při ideálních podmínkách, je ale možné, že se nebude chovat správně při nekorektním zacházení. Nekorektním zacházením by zde mohl být například prázdný seznam cílových bodů. Při kontrole cílových bodů ukládáme splněné podmínky do proměnné, pokud ale proběhne cyklus a nevloží do proměnné žádnou z podmínek, skončí program na *nullPointerException*.

Tomuto problému lze předejít různými způsoby, my zde ale využijeme vzoru *Prázdný objekt*. Vytvoříme takovou proměnnou, která bude vracet validní hodnoty a zároveň nezmění cílové chování programu. Prázdný objekt podmínky zde bude představovat statická třída `NullCondition`.

Pro případ cíle labyrintu, je neutrální chování rovné nesplněné podmínce. Pokud není splněna, je možné pokračovat a nebudou vyhodnocovány ostatní podmínky. Kdybychom logiku změnili a brali žádnou podmínku jako splněnou podmínku, okamžitě po prvním pohybu hráče by se volala metoda kontroly ostatních podmínek. Ta by pouze vyhodnotila cíl jako nesplněný a hru restartovala.

Stručný postup průběhu lekce

1. Vytvořit rozhraní `ICondition`
2. Implementovat rozhraní v třídách `CheckpointCondition` a `FinalTileCondition`
3. Rozšířit třídu `ConditionManager` o seznam koncových bodů bludiště a seznam podmínek pro platný průchod bludištěm
4. Upravit metody pro kontrolu podmínek ve třídě `ConditionManager`, aby upozornily veškeré podmínky o změně stavu a zkontrolovaly jejich plnění
5. Představit problém *nullPointerException*
6. Vytvořit třídu `NullCondition` a využít ji pro předcházení výjimek

5.5.2 Cílový stav

Diagram tříd 4: Tvorba pravidel znázorňuje žlutě třídy nezměněné z minulých lekcí, oranžově již existující třídy upravované touto lekcí a zeleně nově přidané třídy. Řešení lekce zachycuje projekt 5_CreatingRules.

- Lekce přidává rozhraní `ICondition` a třídu `NullCondition`.
- Třídy `CheckpointCondition` a `FinalTileCondition` je třeba upravit, aby vyhovovaly rozhraní `ICondition`
- Třída `ConditionManager` je zásadně upravena, aby používala seznamy podmínek, nikoli proměnné pro jednotlivé podmínky
- Třída `Architect` musí být upravena, aby korektně využívala nově upravenou třídu `ConditionManager`

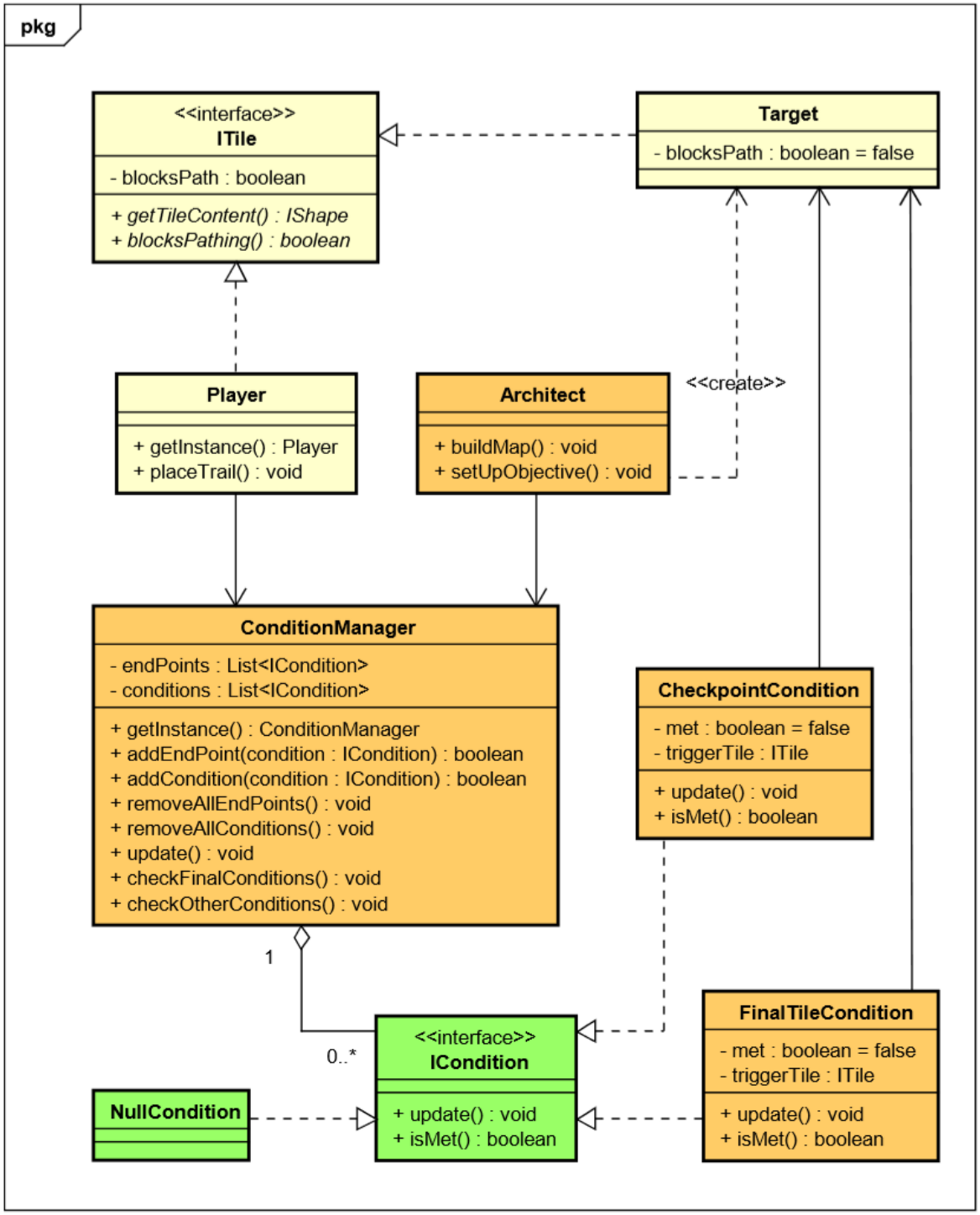

Diagram tříd 4: *Tvorba pravidel*

Projekt 5_CreatingRules obsahuje testovací třídu `L5_CreatingRules`, která ověřuje základní scénáře pro kontrolu splnění lekce. Prvním je, že program restartuje hru po dosažení cíle při nesplnění ostatních podmínek. Další dva scénáře se věnují dosažení obou poskytnutých cílů. Poslední scénář kontroluje, že po selhání hráče je stále možné najít správné řešení a splnit cíl hry.

Poznámky

V lekci je možné přidat další typy podmínek, pro lepší využití rozhraní `ICondition`. V tomto návrhu bylo pro tuto lekci od tohoto kroku odstoupeno, protože seznámení se a práce se seznamy a cykly může být pro studenty náročná sama o sobě.

Další typy podmínek si mohou studenti vymyslet a doplnit ve volném čase. Jejich četnost a variace nepřináší výuce natolik přidanou hodnotu, jak přináší zvýšenou pracnost.

5.6 Lekce 6 – Úprava vyhodnocení konce hry

Lekce pracuje s projektem 5_CreatingRules. Od minulé lekce je možné přidávat dlouhou řadu podmínek, ale pokud pracujeme s malým labyrintem, nemáme příliš místa na smysluplné vkládání jednotlivých prvků. Proto zde upravíme logiku vyhodnocování splnění podmínek, aby bylo možné řešit i takové labyrinty, kde je možné koncovými body procházet.

Chceme zároveň zachovat možnost návratu k původní logice hry, proto zde využijeme vzoru stav pro změnu metody vyhodnocení splnění podmínek. U nově vytvořených tříd nepředpokládáme, že by byly využitelné i v jiných třídách programu, proto je vytvoříme jako vnitřní třídy.

Cíle lekce

- Představit studentům vnitřní třídy
- Vysvětlit a vyzkoušet vzor *Stav*
    - Pro dynamickou změnu funkce programu

5.6.1 Popis lekce

Chceme přesunout rozhodování, jak se zachovat po k výsledku vyhodnocení podmínek, do oddělené metody a mít možnost na začátku, nebo i v průběhu hry měnit, jak se bude tato metoda chovat.

Pro tento účel použijeme vzor *Stav*. Nejprve vytvoříme rozhraní pro sjednocení stavů. Rozhraní je v projektu pojmenováno `IEndGameLogic` a je realizováno jako vnitřní třída pro `ConditionManager`. Pro rozhraní je klíčová pouze funkce `evaluate()`, která provádí rozhodnutí a případné další akce.

Následně vytváříme další vnitřní třídu, `ClosedEnds`, která reprezentuje dosavadní chování programu. Pokud hráč dosáhne jednoho z cílů, vyhodnotí, zda byly podmínky splněny a hráč vyhrál, nebo zda zbývají podmínky ke splnění a hráč začíná hru od znova.

Přidáme také třídu `OpenEnds`, která bude reprezentovat alternativní chování, kde uživatel může procházet cílovými poli, aniž by končila hra. Třídy `OpenEnds` a `ClosedEnds` implementují rozhraní `IEndGameLogic`, proto můžeme do třídy `ConditionManager` přidat proměnnou, která bude uchovávat instanci zvolené logiky.

Přidáme metodu, kterou budeme schopni během testů logiku ovlivňovat, ta je v projektu nazývána `canPassThroughCheckPoints()`. A upravíme metodu `checkOtherConditions()` tak, aby využívala metody vyhodnocení z nového rozhraní `IEndGameLogic`.

Vyzkoušíme, zda stále funguje původní logika, reprezentovaná třídou `ClosedEnds`. Pokud funguje, přepneme logiku na `OpenEnds` a vyzkoušíme její funkčnost. Když budeme postupovat podle metod navržených v testovací třídě `L6_GameEndAlterations`, tak si všimneme, že v současném stavu se podmínky vyhodnotí jako splněné ihned po splnění všech záchytných bodů, pokud uživatel prošel alespoň jedním z cílových bodů.

Aby hra mohla končit pouze v moment, kdy je hráč v cíli, musíme omezit vyhodnocování podmínek pouze na momenty, kdy je hráč přesně na poli cílové podmínky. Doposud jsme pouze kontrolovali, zda je cílové pole překryté, a tudíž dosažené hráčem.

Rozhraní `ICondition` neposkytuje způsob, jakým získat pozici, na kterou se podmínka váže. Musíme tuto možnost do rozhraní přidat. Rozhraní a třídy dané rozhraní implementující nyní musí podporovat metodu `getTriggerTile()`, která vrací `ITile`, ze které již jsme schopni přes `IShape` pozici vyčíst.

Třídu `ConditionManager` rozšíříme o metodu `playerIsOnEnd()`, která nám zkontroluje, zda se právě uživatel nachází na některém z cílových bodů. Vyhodnocení podmínek ukončení hry podmíníme právě pozitivním výstupem nově vytvořené metody.

Závěrem upravíme nastavování cílů v třídě `Architect` a ověříme, že se podmínky vyhodnocují dle našich očekávání a hra může končit pouze na cílových polích.

Stručný postup průběhu lekce

1. Ve třídě `ConditionManager` vytvoříme vnitřní třídu rozhraní `IEndGameLogic` s metodou `evaluate()`
2. Ve třídě `ConditionManager` vytvoříme vnitřní třídu `ClosedEnds`, která implementuje metodu `evaluate()` podle stávající logiky
3. Ve třídě `ConditionManager` vytvoříme vnitřní třídu `OpenEnds`, která implementuje metodu `evaluate()` tak, aby hra nekončila při dosažení prvního konce
4. Aby třída `OpenEnds` mohla vyhodnotit, zda hráč stojí na cíli a nikoli mimo cíl v moment splnění všech podmínek, je nutné rozšířit rozhraní `ICondition` a třídy rozhraní implementující o metodu `getTriggerTile()`

5. Upravíme třídu `OpenEnds` tak, aby hra mohla končit pouze v případě kdy hráč stojí na jednom z konců
6. Upravíme vytvářenou mapu labyrintu a otestujeme správnost chování programu

5.6.2 Cílový stav

Diagram tříd 5: Úprava vyhodnocení konce hry oranžově již existující třídy upravované touto lekcí a zeleně nově přidané třídy. Nově přidávané třídy jsou vnitřními třídami již existující třídy `ConditionManager`. Řešení lekce zachycuje projekt 6_GameEndAlterations.

- Přidáváme rozhraní `IEndGameLogic`
- Přidáváme třídy `OpenEnds` a `ClosedEnds`
- Třída `ConditionManager` je zásadně upravena, aby mohla využívat zvolenou logiku
- Rozhraní `ICondition` a třídy implementující toto rozhraní jsou rozšířeny o metodu `getTriggerTile()`, aby bylo možné zkontrolovat, zda je hráč v cíli, či stále v pohybu
- Třída `Architect` je upravena pro demonstraci a testování nového režimu vyhodnocování podmínek

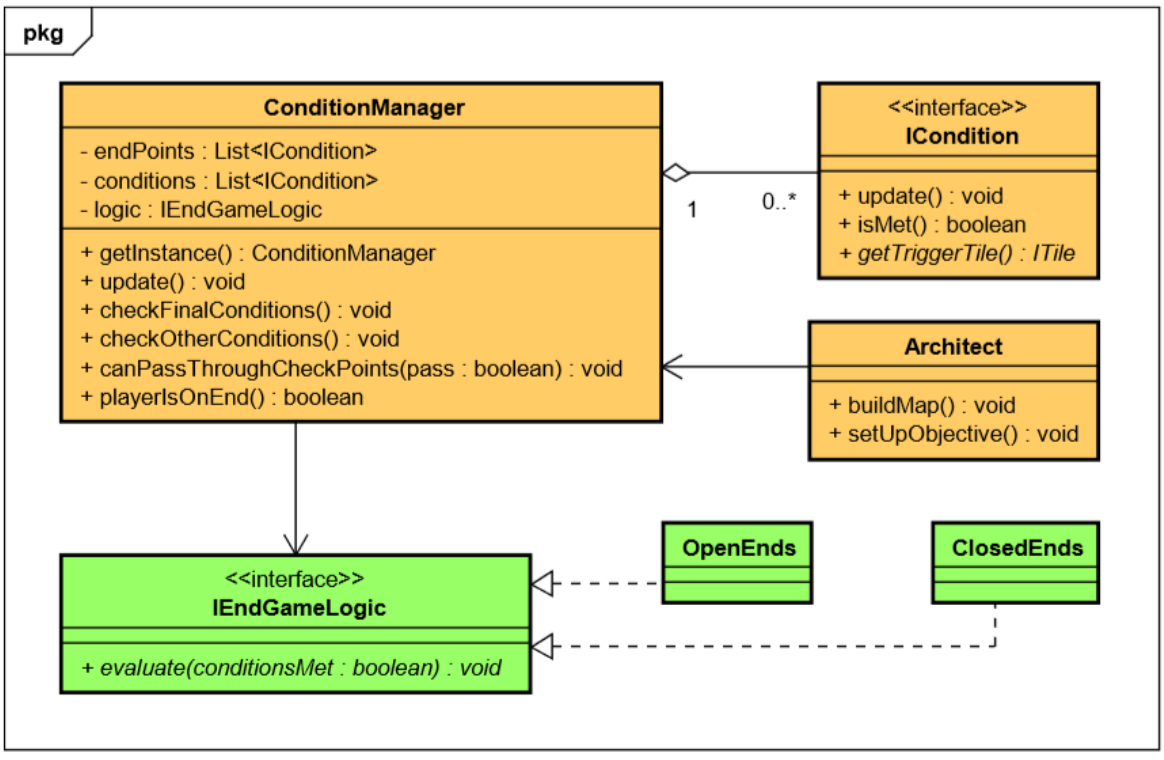

Diagram tříd 5: *Úprava vyhodnocení konce hry*

Po dokončení lekce, lze ověřit za pomoci testovací třídy L6_GameEndAlterations, že hra podporuje následující scénáře. Prvním scénářem je, že hráč projde veškeré záchytné body, ale v průběhu projde všemi cílovými poli. Hra se neukončí a nelze dosáhnout cíle.

Druhý scénář prověřuje přepnutí na uzavřené konce, že po dosažení prvního cíle, se hra resetuje a je možné pokračovat v jejím hraní. Třetí scénář kontroluje, že lze hru úspěšně dokončit při otevřených průchodech, tedy když lze volně procházet cíli.

Poznámky

Pro studenty, aby bylo určitě velmi zajímavé přesunout kontrolu, zda je hráč v cílovém poli, mezi ostatní podmínky, aby se jednalo i z logického pohledu o podmínku, nikoli o samostatnou metodu. To by ale vyžadovalo značnou úpravu rozhraní `ICondition` a bylo by nutné zpřístupnit seznam cílových bodů samotné podmínce.

Není vhodné v této fázi výrazně zvyšovat komplexitu programu, proto není vhodné zavádět podobnou podmínku v této lekci. Bude efektivnější takové a další podmínky přidávat v rámci samostudia studentů, nebo v rozšiřující lekci. Takovému rozšíření se tato práce nevěnuje.

5.7 Shrnutí

Kapitola návrhu postupu výuky rozděluje projekt postupně vylepšovaného programu do šesti lekcí, ke kterým je vytvořen doprovodný program. Doprovodný program je generátor projektů, které reprezentují počáteční a cílové stavy lekcí.

Další mezi stavy nejsou v rámci této práce zachyceny. Důvodem je, že doprovodný program slouží jako reprezentace a průkaz přístupu, není určen pro podporu výuky. Pro výuku by měl být vytvořen nový projekt s rozdělením lekcí s ohledem na prostředí v jakém bude použita, což se projeví zejména na podrobnosti a rozdělení lekcí do cvičení.

Návrh tak jak je uveden v této práci neprošel iteracemi revizí, je proto vhodné při jeho implementaci věnovat pozornost průběhu a výsledkům výuky. Nad konkrétní implementací je vhodné zaznamenávat nastalé problémy, pozorovaná rizika a na závěr každého z cyklů analyzovat, jak se lze z této zkušenosti ponaučit. Ideálně by další cyklus výuky měl aplikovat opatření pro minimalizaci rizik, odstranění problémů a zlepšení příkladu.

6 Závěr

Práce uvedla komentovanou rešerši zdrojů, na kterých popisuje současný stav problematiky. Problematika výuky programování byla rozdělena do oblastí výuky objektově orientovaného programování, výuky návrhových vzorů, výuky počítačových věd a výuky obecně. Také byla přidána sekce podpory výuky jak nástroji, tak metodikami a gamifikací.

Ukázalo se, že zkušenosti studentů mohou mít negativní vliv na jejich schopnost učit se některé koncepty a je proto dobré organizaci výuky jejich potřebám přizpůsobit. Také se ukázalo, že podpora zájmu skrze gamifikaci je často neúspěšná a její popularita v posledních letech klesá.

6.1 Dosažení vytyčených cílů

Zpracování práce vyústilo v následující:

- Byla provedena analýza námětů používaných ve vstupních kurzech programování. Analýza byla provedena metodou systematické rešerše literatury. Seznam používaných námětů byl rozšířen o použitelné náměty, zmiňované ve zkoumaných zdrojích.
- Byl navržen námět pro program, který by bylo možné použít při výuce podle metodiky Architecture First, inspirovaný známou hrou, hledání průchodu bludištěm.
- Byl navržen postup výuky při postupném rozpracovávání výše navrženého námětu. Pro generování postupně zdokonalovaných projektů se využívá program Monolith2Projects.

6.2 Možnosti rozšíření práce

Na práci je možné navazovat:

- Podrobnou analýzou pedagogických vzorů a jejich použití v navržené výuce.
- Rozpracováním návazné výuky na vytyčené lekce.
- Rozšířením výuky o základy kontroly kvality softwaru, tedy analýzou cílů testování, návrhem testů a vlastním testováním.

Příloha A: Přehled návrhových vzorů

Použité vzory

- **Přepravka** (Crate) [39] – Používá jeden objekt, který představuje množinu různých hodnot.
- **Jedináček** (Singleton) [58] – Zajišťuje, aby byla vytvořena pouze jedna instance třídy a poskytuje globální přístupový bod k objektu.
- **Knihovna** (Library / Utility class) [42] – Třída obvykle obsahující statické metody obecného určení.
- **Originál** (Multiton) [21] – Zajišťuje, aby dvě instance třídy nebyly shodné při srovnávání. Poskytuje globální přístup k instancím. Seznam instancí lze dynamicky měnit.
- **Prázdný objekt** (Null Object) [58] – Poskytněte objekt jako náhradu za nedostatek objektu daného typu. Prázdný objekt poskytuje inteligentní chování „žádného chování“.
- **Prostředník** (Mediator) [58] – Definuje objekt, který zapouzdřuje, jak sada objektů interaguje. Podporuje volné spojení tím, že vede objekty k tomu, aby se navzájem explicitně odvolávaly, a dovoluje jejich vzájemné nezávislé ovlivňování.
- **Služebník** (Servant) [39] – Poskytuje společné funkce pro sadu tříd.
- **Stavitel** (Builder) [58] – Odděluje konstrukci komplexního objektu od jeho reprezentace a umožnuje používat stejný konstrukční proces pro tvorbu různých reprezentací.
- **Tovární metoda** (Factory method) [58] – Definuje rozhraní pro vytvoření jediného objektu, ale nechává podtřídy rozhodnout, která třída se má instancovat. Umožňuje třídě odložit instancování na podtřídy.
- **Výčtový typ** (Enum) [58] – Zajišťuje, že třída má pouze pojmenované instance a poskytuje k nim globální přístup. Seznam instancí je předen určen a nelze měnit.
- **Stav** (State) [39] – Umožňuje objektu změnit jeho chování při změně interního stavu.

Zmiňované vzory

- **Adaptér** (Adapter) [58] – Převádí rozhraní jedné třídy do rozhraní jiné třídy. Adaptér umožňuje třídám s rozdílným rozhraním pracovat společně.
- **Dekorátor** (Decorator) [58] – Připojuje další odpovědnosti k objektu, přičemž udržuje stejné rozhraní. Dekorační prvky poskytují flexibilní alternativu k podtřídě pro rozšíření funkčnosti.

- **Fasáda** (Facade) [15] – Zajišťuje jednotné rozhraní pro sadu rozhraní v subsystému. Definuje rozhraní vyšší úrovně, které usnadňuje používání subsystému.
- **Interpret** (Interpreter) [58] – V daném jazyku definujte reprezentaci gramatiky, interpret používá reprezentaci k interpretaci vět v jazyce.
- **Iterátor** (Iterator) [58] – Zajistěte způsob, jak postupně přistupovat k prvkům agregovaného objektu, aniž by byla potřeba znát jeho podkladovou reprezentaci.
- **Líná inicializace** (Lazy initialization) [4] – Taktika zpoždění tvorby objektu, výpočtu hodnoty nebo nějakého jiného drahého procesu, dokud není potřeba poprvé. Také označován jako „virtuální proxy“.
- **Model-Pohled-Ovládání** (Model – View – View model) [9] – Rozděluje logiku a uživatelské rozhraní programu na samostatné entity a propojuje je prostřednictvím modelu zobrazení.
- **Most** (Bridge) [39] – Odděluje abstrakci od implementace, aby se mohly nezávisle lišit.
- **Muší váha** (Flyweight) [58] – Používá sdílení pro podporu velkého počtu objektů, které mají společný stav, kde se může lišit jiná část stavu.
- **Návštěvník** (Visitor) [58] – Představuje operaci, která má být provedena na prvcích struktury objektu. Návštěvník umožňuje definovat novou operaci bez změny tříd prvků, na kterých pracuje.
- **Pamětník** (Memento) [58] – Zachycuje vnitřní stav objektu, aniž by došlo k narušení zapouzdření, a tím poskytnete způsob pro obnovení objektu do počátečního stavu v případě potřeby.
- **Pozorovatel** (Observer) [39] – Definuje závislost typu jeden na více mezi objekty, ve kterých změna stavu v jednom objektu vede k automatickému oznámení a aktualizaci všech jeho závislých.
- **Prototyp** (Prototype) – Určuje typy objektů, které chceme vytvořit pomocí prototypové instance, a vytváří nové objekty z kostry stávajícího objektu.
- **Přední ovladač** (Front controller) [14] – Vzor se týká návrhu webových aplikací. Poskytuje centralizovaný vstupní bod pro zpracování požadavků.
- **Strom** (Composite) [58] – Sestavuje objekty do stromových struktur, které reprezentují dílčí hierarchie. Umožňuje zacházet jednotlivě se samostatnými objekty a složením objektů.
- **Šablonová metoda** (Template method) [58] – Definujte kostru algoritmu v operaci, odkláněním některých kroků do podtříd. Metoda šablony umožňuje podtřídám předefinovat určité kroky algoritmu bez změny struktury algoritmu.

Příloha B: Přehled pedagogických vzorů [28]

Vzory přímé interakce (LEA1)

- **Část mysli** (Piece of Mind) – Na konci výuky nechte studenty poskytnout anonymní zpětnou vazbu, abyste znali jejich vnímání obtížných částí.
- **Dobrý posluchač** (Uninterrupted Listening) – Nechte studenty dokončit svou odpověď, když mluví, abyste pochopili jejich plné myšlení a mohli na to správně reagovat.
- **Dramatická pauza** (Pregnant Pause) – Dejte všem studentům dostatek času na formulování odpovědi na vaši otázku. Pomlka může také zvýšit napětí a přitáhnout pozornost studentů k dané otázce.
- **Jednoduché odpovědi** (Simple Answer) – Zapojte studenty do jednoduchých otázek s uzavřenými otázkami, jejichž odpovědi lze snadno určit z materiálu, který jste již ve třídě pokryli.
- **Komentovaná ukázka** (Commented Action) – Uveďte popis kroků, které provádíte při předvádění složitých operací, vystavte studenty sémiotické doméně.
- **Minimální odstup** (Minimum Distance) – Uzavřete fyzické oddělení mezi vámi a studenty pohybem v místnosti.
- **Nikdo není dokonalý** (Nobody Is Perfect) – Nesnažte se být dokonalí a přiznejte svá omezení ušlechtile.
- **Oceňování otázek** (Honor Questions) – Motivujte studenty k tomu, aby kladli otázky, ukažte jim, že si jich ceníte a že nejsou žádné hloupé otázky.
- **Odůvodnění** (Line of Reasoning) – Pokud jsou odpovědi studentů nečekané, použijte je pro učení. Požádejte studenta, aby vysvětlil svůj proces myšlení a analyzujte odpověď.
- **Otevřené otázky** (Open-Ended Questions) – Vypracujte otevřené otázky před třídou.
- **Odkazování se na program** (Reference the Plan) – Nechte studenty vědět, že máte plán pro celkový kontext a kde se v něm právě nachází.
- **Pečlivě připravené otázky** (Carefully Crafted Questions) – Věnujte čas před výukou, abyste vytvořili otázky, které pomáhají dosáhnout požadovaných cílů lekce.
- **Role jazykového modelu** (Language Role Model) – Pokud vyučujete obsah v cizím jazyce, ujistěte se, že ho umíte používat správně, protože studenti vás pravděpodobně budou napodobovat. Souvisí se vzorem Řeč učitele, ale zaměřuje se na výuku v cizím jazyce.

- **Řeč těla** (Body Language) – Podporujte své mluvené slovo jazykem těla pomocí gest a výrazů obličeje záměrně.
- **Řeč učitele** (Teacher's Language) – Mluvte nahlas a používejte jazyk, který vyhovuje účastníkům (nebo studentům) a zároveň je jak výrazný, tak fascinující.
- **Studenty řízená přednáška** (Student Driven Lecture) – Zvolte na začátku přednášky otázky, na které studenti dnes chtějí odpovědi nejvíce a revidujte (části) své přednášky. Řešení by mohlo být aplikováno pomocí sociálních médií nebo e-learningových zařízení. Dřívější předložené domácí úkoly by mohly tvořit i základnu (nepřímo) pro studenty řízenou přednášku.
- **Vítání účastníků** (Welcome the Participants) – Přivítejte účastníky semináře (nebo studenty v přednášce), promluvte s nimi neformálně na začátku, čímž vytvoříte vhodnou atmosféru. Je to důležitější pro semináře, ale na začátku kurzu bychom měli také vítat studenty a uplatňovat některé aspekty tohoto modelu.

Vzory chování v průběhu lekce (LEA2)

- **Barevná analogie** (Colorful Analogy) – Použijte barevné analogie, abyste představili koncept, který má spoustu nudných a detailních následků. Poskytuje místo, kam se můžete vrátit, abyste získali detaily. Vztahuje se ke stimulaci představivosti a vzoru Konzistentní metafory.
- **Diagnostika a prevence nedostatků** (Pitfall Diagnosis and Prevention) – Věnujte zvláštní pozornost důležitým konceptům a zdůrazněte je, pokud se ukáže, že by s tím studenti měli problémy.
- **Experimentace** (Take a Risk) – Vyzkoušejte nové věci v pedagogice, pokud nevíte, zda to zafunguje, ujistěte se, že máte zálohu s alternativními a pracovními metodami. Získejte zpětnou vazbu a použijte ji ke zlepšení. Hlavním poselstvím je, že je třeba experimentovat s různými formami doručování informací a následně vyhodnotit jejich účinnost.
- **Fyzická analogie** (Physical Analogy) – Použijte, je-li to možné, konkrétní způsob, jak ilustrovat dynamické vlastnosti abstraktního konceptu.
- **Jeden koncept, více implementací** (One Concept – Several Implementations) – Použijte různé implementace jako příklady jednoho abstraktního konceptu a poté je porovnávejte, abyste objevili podstatu abstraktního konceptu.
- **Konzistentní metafora** (Consistent Metaphor) – Použijte metaforu, která je v souladu s vyučovaným tématem, včetně stejných základních prvků a jejich interakcí. Vztahuje se k Barevné analogii.
- **Média účastníků** (Use Participant's Media) – Podpořte studenty, aby si z prezentace ukládali poznámky nebo kopírovali věci tím, že používáte média stejného druhu jako média studentů. Ačkoli je název v dnešním prostředí poněkud zavádějící (může být

interpretován jako sociální média nebo přístroje), obsah je jistě použitelný pro přednášky: Pokud jsou studenti povinni kopírovat věci ručně, pak je zapište nebo je také natočte ručně.

- **Myslet, diskutovat, sdílet** (Think..Pair..Share) – Aktivně zapojíte studenty tím, že je (1) necháte odpovědět na otázku osobně, pak (2) diskutovat o tom s jiným studentem a (3) náhodně vybrané studenty necháte sdílet s třídou odpověď svého partnera. Nemusí dobře fungovat ve větších skupinách studentů
- **Nadšení učitele** (Teacher's Enthusiasm) – Zvyšte nadšení studentů tím, že projevíte nadšení sami k předmětu, ke studentům a k učení. Toto je samozřejmě obecně použitelné, ale zvláště důležité v přednáškách, jelikož učitel v nich hraje ústřední roli. Motivace studentů k učení je jednou ze základních funkcí přednášek a nadšení je velmi důležitým motivátorem
- **Nárazníky** (Buffers) – Zahrnujte nárazníky do plánování přednášky, abyste měli čas reagovat na nepředvídané problémy, otázky a témata.
- **Nechme plán plavat** (Let the Plan Go) – Pokud se vyskytne otázka, kterou jste neočekávali při plánování, ale odpověď by vaši přednášku vylepšilo, pak se otázce věnujte. To vyžaduje, abyste do plánování zahrnuli nárazníky.
- **Nechte je udělat z toho jejich problém** (Make Them Make It Their Problem) – Nechte studenty navrhnout výukové sekce, kde budou muset aplikovat zásady vzoru Udělejte z toho jejich problém.
- **Odhalení procesu** (Expose the Process) – Neukazujte pouze výsledky a správné řešení, ale vysvětlete také proces, jak se dobrat výsledku, včetně kritických rozhodovacích bodů.
- **Opakování** (Repeat Yourself) – Opakujte důležitá témata, a také je propojte s novými. To pomůže studentům si látku zapamatovat.
- **Praktická ukázka** (Show It Running) – Neříkejte jen o funkčnosti softwaru, ale také jej využijte během prezentace, neboť si studenti zapamatují lépe, pokud viděli, jak pracuje, a vy můžete poukazovat na kritické a důležité části.
- **Propojení starého s novým** (Linking Old to New) – Pomozte studentům, aby propojili nové informace s existujícími poznatky pomocí starého obalu, který představí tyto nové informace. Vztahuje se k Rozšiřování známého světa.
- **První přítomný** (Be There First) – Být v učebně před studenty dává čas na přípravu přednášky a osobní komunikaci s přicházejícími studenty.
- **Přestávky** (Breaks) – Zahrňte (pravidelné) přestávky, pokud jsou relace (nebo přednášky) velmi dlouhé nebo intenzivní. Další možností pro dlouhé přednášky je pravidelné upoutávání pozornosti.

- **Relevantní příklady** (Relevant Examples) – Použijte příklady pro ilustraci abstraktních pojmů. Tyto příklady mohou souviset s každodenním životem studentů, např. související s hudbou nebo sporty.
- **Různé přístupy** (Different Approaches) – Přistupujte k tématu různými přístupy, abyste zohlednili různé způsoby vnímání. Přístupy vhodné pro přednášky jsou určitě jen podmnožinou všech stávajících. Výběr vhodného způsobu výkladu pomáhá při provádění tohoto výběru.
- **Řešení před abstrakcí** (Solution Before Abstraction) – Nechte studenty nejprve nalézt řešení konkrétních problémů, nechte je identifikovat společné aspekty těchto řešení a použijte tyto identifikované aspekty, abyste představili obecný abstraktní koncept. Je to v podstatě v rozporu se vzorem Nejprve obecné, ale oba se zabývají různými styly učení (holistický vs. serializmus). Proto by se měly používat oba nebo v kombinaci.
- **Simulace** (Simulation Games) – Simulace složité činnosti pomáhá lépe pochopit základní koncepty než jednoduché vysvětlení, a navíc nabízí příležitost k interakci. Použitelné pouze v malých skupinách a s dobrou přípravou.
- **Stravitelné celky** (Digestible Packets) – Uspořádejte témata přednášky tak, aby zůstaly malé a srozumitelné a mohly být dokončeny v přiměřeném čase. V přednáškách často není možné nebo žádoucí mít více přestávek. V takovém případě vám může pomoct upoutávání pozornosti.
- **Udělejte z toho jejich problém** (Make It Their Problem) – Nechte studenty řešit problém a shodnout se na jeho řešení a nechte je nasměrovat vás k pochopení řešení pomocí jednoho prezentačního počítače.
- **Ukázka programování** (Show Programming) – Ukažte programování v akci (a ne pouze snímky o programování), vysvětlete kód a pojmy, které používáte, a ukažte, jak používat užitečné funkce IDE.
- **Ukázat, pak vysvětlit** (See Before Hear) – Poskytněte žákům příležitost vidět a prožít nový koncept před tím, než se o něm dozví.
- **Zkuste si sami** (Try It Yourself) – Po představení nového konceptu přejděte na prezentaci a nechte studenty absolvovat cvičení, které využívá dříve představené koncepty. Zkontrolujte, zda to pochopili a zda poskytují zpětnou vazbu. Používá se pouze v menších třídách.
- **Změny média** (Change Media) – Diverzifikujte přednášku pomocí různých médií a pravidelně je měňte (neměňte je ale příliš často). Volba dostupných médií může být externě určena. Souvisí s výběrem vhodné formy výuky a strukturováním lekcí.
- **Známé příklady** (Acquaintance Examples) – Vyberte si příklady, se kterými jsou studenti seznámeni, ale které nejsou v oblasti odborných znalostí studentů.

- **Zpětná vazba** (Feedback) – Dejte studentům zpětnou vazbu na jejich výkon, aby věděli, kde chybují. Učení se pak stává kompletnějším. Zpětná vazba může být poskytnuta na přednášce, ale pouze pokud je relevantní pro celou skupinu. Existuje několik vzorů o sdílení zpětné vazby

Vzory výuky problematiky (LEA3)

- **Abstrakce shora dolů** (Abstraction Gravity – From High to Low) – Představte koncepty, které musí být pochopeny na dvou úrovních abstrakce nejprve na jejich nejvyšší úrovni, a pak je propojit s nižší úrovní abstrakce pomocí reflexe. Podobnosti se vzorem Nejprve obecné.
- **Nejprve obecné** (General Concepts First) – Začněte s výukou obecných konceptů, aby si studenti mohli zapamatovat na nová, podrobnější témata, která se jim učí snadněji později, protože mohou spojovat tato témata s dříve naučenými obecnými pojmy. Podobnosti se vzory Abstrakce shora dolů a První vlaštovka. V některých případech je lepší začít s konkrétními příklady.
- **Orientace na problém** (Problem Orientation) – Představte nové téma ukázkou řešení problému, ať studenti ví, kam je vedete. Použití problému jako motivátor obsahu je určitě doporučeno, ale existují také argumenty, proč nekombinovat s přístupem shora dolů, protože by se to nemuselo hodit různým stylům učení studentů. Zdá se, že kombinace obou přístupů (shora dolů a zdola nahoru) funguje dobře.
- **Pojmenovat na konec** (Name is Last) – Zaměřte se nejdříve na pochopení tématu nebo konceptu před uvedením jeho jména.
- **Příběh modulu** (Module's Story) – Proveďte tok modulu (nebo přednášky) do příběhu pomocí příkladu, cvičení nebo cíle, který využívá většinu nebo všechny témata v modulu.
- **Rozdělení podobné látky** (Separate Similar Content) – Jsou-li témata různá, ale zdají-li se podobná, pak časově rozdělujte jejich pokrytí a vysvětlete za jakých podmínek se která používají.
- **Rozšiřování známého světa** (Expand the Known World) – Výslovně propojte nový koncept se zkušenostmi studentů. Spojování lze provést např. Barevnou analogií nebo Konzistentní metaforou. Souvisí se vzorem Propojení starého s novým.
- **Řízená diskuse** (Discussion with Peers and Staff) – Poskytněte studentům příležitost zapojit se do diskuse o nápadech, s nimiž se setkávají, a informovat je o svých očekáváních, jak chcete, aby se věci diskutovali. Ačkoli je primárně určen pro e-learning, tento vzor může být použit i v běžných lekcích.
- **Sedm částí** (Seven Parts) – 7 se zdá být dobrým počtem kroků nebo dílčích témat obsažených v modulu. Velmi závisí na délce přednášky a dalších možnostech interaktivity.

- **Shrnutí** (Summary) – Poskytněte shrnutí na konci každého zasedání nebo přednášky, které opakuje pokrytá důležitá témata a jejich souvislost s celkovým obsahem. V přednášce se často objevují i cíle učení, které jsou uvedeny v souhrnu
- **Spirála** (Spiral) – Začněte s uvedením propojených témat na vysoké úrovni tak, aby je studenti mohli používat dříve, než začnou pracovat na smysluplných a zajímavých problémech. Iterativně pokrýt témata podrobněji pak.
- **Širší perspektiva** (Wider Perspective) – Dát studentům interdisciplinární pochopení důsledků technologie (např. ekonomického, sociálního nebo organizačního) tím, že učí technické téma z širší perspektivy.
- **Útok na více frontách** (Multi-Pronged Attack) – Vyberte si příklady a cvičení, které pokrývají více než jednu myšlenku nebo téma současně.
- **Vše připraveno** (Prepare Equipment) – Ujistěte se, že při zahájení lekce nebo přednášky je vaše zařízení funkční a připravené k použití.

Vzory vedení kurzu (LEA4)

- **Aktivní student** (Active Student) – Udržujte studenty aktivní, protože pasivní studenti, kteří pouze poslouchají a čtou, se moc nenaučí. To platí také pro přednášky. Otázky jsou dobrým způsobem udržení pozornosti, jsou řešeny několika vzory.
- **Harmonogram** (Seminar Plan) – Mějte plán nebo program semináře, který zdůrazňuje důležitá témata a cíle a definuje vaši strategii a pořadí jejich pokrytí.
- **Iterativní zlepšování kurzu** (Iterative Course Development) – Zlepšujte kurz (a přednášky) na základě předchozích zkušeností iterativně. Původně určen pro celý kurz, ale je použitelný pro přednášku nebo lekci.
- **Kombinace online a tváří v tvář komunikací** (Integrated F2F and Online Activities) – Máte-li také online výuku, integrujte ji s aktivitami tváří v tvář, aby se navzájem doprovázely a vzájemně doplňovaly a podporovaly studijní dovednosti. Souvisí s výběrem vhodného obsahu a formy.
- **Kontrola předpokladů** (Check Prerequisites) – Ujistěte se před výukou, že vše, co potřebujete je k dispozici a funguje. Vztahuje se k volbě vhodného obsahu a formy výuky.
- **Nový přístup k novému tématu** (New pedagogy for new paradigms) – Použijte pedagogiku, která odpovídá způsobům myšlení požadovaným v paradigmatu, kterou vyučujete. Určeno pro programování, ale pravděpodobně obecně použitelné.
- **První vlaštovka** (Early Bird) – Učte nejdůležitější témata jako první v kurzu, neboť si studenti nejlépe pamatují to, co se nejdříve učí. Podobné jako vzor Nejprve obecné.

- **Příprava scény** (Set the Stage) – Připravte studenty před zavedením nového materiálu tak, že prohlédnete předpoklady, ukážete cíl, kontext a vymezení.
- **Řád země** (Lay of the Land) – Představte studentům brzy v průběhu kurzu velký artefakt, který pokrývá hlavní předměty, nechte je prozkoumat, aby věděli, kam směřuje kurz, a lépe umístit podrobnosti, které budou pokryty později. Artefakt může být také velkým příkladem nebo analogií.
- **Skripta** (Manuscript) – Poskytněte skripta, která doplňují přednášky, a to explicitně uvedením důležitých cílů, faktů nebo milníků. Úzce souvisí s výběrem vhodného obsahu.

Použité informační zdroje

[1]. ADAMSON Chris. *Query by Slice, Parallel Execute, and Join: A Thread Pool Pattern* | Oracle Community. [online]. 2008 [cit. 2018-03-24]. Dostupné z: https://community.oracle.com/docs/DOC-983191

[2]. BENNEDSEN, Jens., Michael E. CASPERSEN a Michael. KÖLLING. *Reflections on the teaching of programming: methods and implementations*. New York: Springer, c2008. Lecture notes in computer science, 4821. ISBN 978-3-540-77933-9.

[3]. BERGIN, Joseph. *Pedagogical patterns: advice for educators*. Compact ed., 2., (corrected) print. Pleasantville, NY: Joseph Bergin Software Tools, 2012. ISBN 9781479171828.

[4]. BISHOP Philip a Nigel WARREN. *Java Tip 67: Lazy instantiation – Balancing performance and resource usage.* JavaWorld, 1999 [cit. 2018-03-24]. Dostupné z: https://www.javaworld.com/article/2077568/learn-java/java-tip-67--lazy-instantiation.html

[5]. BÖRSTLER, Jürgen, Marie NORDSTRÖM a James H. PATERSON. *On the Quality of Examples in Introductory Java Textbooks*. ACM Transactions on Computing Education [online]. 2011, 11(1), 1-21 [cit. 2018-03-24]. DOI: 10.1145/1921607.1921610. ISSN 19466226. Dostupné z: http://dl.acm.org/citation.cfm?doid=1921607.1921610

[6]. CARRO, Manuel, Ángel HERRANZ a Julio MARIÑO. *A model-driven approach to teaching concurrency.* ACM Transactions on Computing Education [online]. 2013, 13(1), 1-19 [cit. 2018-03-24]. DOI: 10.1145/2414446.2414451. ISSN 19466226. Dostupné z: http://dl.acm.org/citation.cfm?doid=2414446.2414451

[7]. COOPER, Stephen, Wanda DANN a Randy PAUSCH. *Teaching objects-first in introductory computer science*. ACM SIGCSE Bulletin [online]. 2003, 35(1), 191- [cit. 2018-03-24]. DOI: 10.1145/792548.611966. ISSN 00978418. Dostupné z: http://portal.acm.org/citation.cfm?doid=792548.611966

[8]. DOUGHERTY Chad, SAYRE Kirk, SVOBODA David, TOGASHI Kazuya a Robert C. SEACORD. *Secure Design Patterns*. Carnegie Mellon University – Software Engineering Institute [online]. 2009 [cit. 2018-03-24]. Dostupné z: https://resources.sei.cmu.edu/asset_files/TechnicalReport/2009_005_001_15110.pdf

[9]. DUŠEK Filip. *A Technological View on Modern UI in Games*. Game Developer Session 2016, Praha, 4.-5. listopad, 2016

[10]. EVANS, Eric. *Domain-driven design: tackling complexity in the heart of software.* Boston: Addison-Wesley, c2004. ISBN 978-0321125217.

[11]. FAIN, Yakov. *Java programming 24-hour trainer, second edition.* 2nd edition. Indianapolis, IN: John Wiley, 2015. ISBN 978-1-118-95145-3.

[12]. FINCHER, Sally, Stephen COOPER, Michael KÖLLING a John MALONEY. *Comparing alice, greenfoot & scratch.* In: Proceedings of the 41st ACM technical symposium on Computer science education – SIGCSE '10 [online]. New York, New York, USA: ACM Press, 2010, 2010, s. 192- [cit. 2018-03-24]. DOI: 10.1145/1734263.1734327. ISBN 9781450300063. Dostupné z: http://portal.acm.org/citation.cfm?doid=1734263.1734327

[13]. FLEURY, Ann E. *Encapsualtion and reuse as viewed by java students*. In: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education – SIGCSE '01 [online]. New York, New York, USA: ACM Press, 2001, 2001, s. 189-193 [cit. 2018-03-24]. DOI: 10.1145/364447.364582. ISBN 1581133294. Dostupné z: http://portal.acm.org/citation.cfm?doid=364447.364582

[14]. FOWLER, Martin. *Patterns of enterprise application architecture*. Boston: Addison-Wesley, c2003. ISBN 978-0-321-12742-6.

[15]. FREEMAN, Eric, Elisabeth. ROBSON, Kathy. SIERRA a Bert. BATES. *Head First design patterns.* Sebastopol, CA: O'Reilly, c2004. ISBN 978-0596007126.

[16]. GAMMA, Erich. *Design patterns: elements of reusable object-oriented software.* Reading, Mass.: Addison-Wesley, c1995. ISBN 0-201-63361-2.

[17]. GESTWICKI, Paul a Fu-Shing SUN. *Teaching Design Patterns Through Computer Game Development.* Journal on Educational Resources in Computing [online]. 2008, 8(1), 1-22 [cit. 2018-03-24]. DOI: 10.1145/1348713.1348715. ISSN 15314278. Dostupné z: http://portal.acm.org/citation.cfm?doid=1348713.1348715

[18]. GÓMEZ-MARTÍN, Marco Antonio, Guillermo JIMÉNEZ-DÍAZ a Javier ARROYO. *Teaching design patterns using a family of games.* In: Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education – ITiCSE '09 [online]. New York, New York, USA: ACM Press, 2009, 2009, s. 268- [cit. 2018-03-24]. DOI: 10.1145/1562877.1562960. ISBN 9781605583815. Dostupné z: http://portal.acm.org/citation.cfm?doid=1562877.1562960

[19]. GRAND, Mark. *Patterns in Java: a catalog of reusable design patterns illustrated with UML*. 2nd ed. Indianapolis, Ind.: Wiley Pub., 2002. ISBN 978-0471227298.

[20]. HADJERROUIT, Said. *A constructivist framework for integrating the Java paradigm into the undergraduate curriculum*. ACM SIGCSE Bulletin [online]. 1998, 30(3), 105-107 [cit. 2018-03-24]. DOI: 10.1145/290320.283079. ISSN 00978418. Dostupné z: http://portal.acm.org/citation.cfm?doid=290320.283079

[21]. HOULE Paul. *Generation 5 »The Multiton Design Pattern* [online]. 2008 [cit. 2018-03-24]. Dostupné z: http://gen5.info/q/2008/07/25/the-multiton-design-pattern/

[22]. HUBWIESER Peter. *Computer Science Education in Schools Special Issue of the ACM Journal "Transactions on Computing Education" Solicitation Letter*. TOCE ACM [online]. February 2013 [cit. 2018-03-24]. Dostupné z: https://toce.acm.org/announcements.cfm

[23]. JAMES PAUL GEE. *What video games have to teach us about learning and literacy.* New York, NY [u.a.]: Palgrave Macmillan, 2004. ISBN 1403965382.

[24]. JOHNSON, Claire. *Learning to program with Game Maker.* International Journal of Computer Science Education in Schools [online]. 2017, 1(2), 17- [cit. 2018-03-24]. DOI: 10.21585/ijcses.v1i2.5. ISSN 2513-8359. Dostupné z: http://www.ijcses.org/index.php/ijcses/article/view/5

[25]. KASURINEN, Jussi a Antti KNUTAS. *Publication trends in gamification: A systematic mapping study. Computer Science Review* [online]. 2018, s. 27, 33-44 [cit. 2018-03-24]. DOI: 10.1016/j.cosrev.2017.10.003. ISSN 15740137. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S1574013716301769

[26]. KNUTAS, Antti. *Profile-Based Algorithm for Personalized Gamification in Computer-Supported Collaborative Learning Environments*. ResearchGate | Share and discover research [online]. Copyright © by the paper [cit. 2018-03-24]. Dostupné z: https://www.researchgate.net/publication/320387170

[27]. KÖPPE, Christian a Joost SCHALKEN-PINKSTER. *Lecture design patterns.* In: Proceedings of the 18th European Conference on Pattern Languages of Program – EuroPLoP '13 [online]. New York, New York, USA: ACM Press, 2015, 2015, s. 1-27 [cit. 2018-03-24]. DOI: 10.1145/2739011.2739015. ISBN 9781450334655. Dostupné z: http://dl.acm.org/citation.cfm?doid=2739011.2739015

[28]. KÖPPE, Christian. *Towards a pattern language for lecture design*. In: Proceedings of the 18th European Conference on Pattern Languages of Program – EuroPLoP '13 [online]. New York, New York, USA: ACM Press, 2015, 2015, s. 1-17 [cit. 2018-03-24]. DOI: 10.1145/2739011.2739014. ISBN 9781450334655. Dostupné z: http://dl.acm.org/citation.cfm?doid=2739011.2739014

[29]. LAKER Pete. *Blackboard Design Pattern* – TechNet Articles – United States (English) - TechNet Wiki. Document Moved [online]. Copyright © 2015 Microsoft Corporation. All rights reserved. [cit. 2018-03-24]. Dostupné z: https://social.technet.microsoft.com/wiki/contents/articles/13215.blackboard-design-pattern.aspx

[30]. LEA, Douglas. *Concurrent programming in Java: design principles and patterns*. 2nd ed. Reading, Mass.: Addison-Wesley, c2000. ISBN 0-201-31009-0.

[31]. LINDBERG, Renny S. N., Aziz HASANOV a Teemu H. LAINE. *Improving Play and Learning Style Adaptation in a Programming Education Game.* In: Proceedings of the 9th International Conference on Computer Supported Education [online]. SCITEPRESS – Science and Technology Publications, 2017, 2017, s. 450-457 [cit. 2018-03-24]. DOI: 10.5220/0006350304500457. ISBN 978-989-758-239-4. Dostupné z: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006350304500457

[32]. MAYER, Richard E. *The Psychology of How Novices Learn Computer Programming.* ACM Computing Surveys [online]. 13(1), 121-141 [cit. 2018-03-24]. DOI: 10.1145/356835.356841. ISSN 03600300. Dostupné z: http://portal.acm.org/citation.cfm?doid=356835.356841

[33]. MINER, Donald. a Adam SHOOK. *MapReduce design patterns*. Sebastopol, CA: O'Reilly, 2013. ISBN 1449327176.

[34]. MURATET, *Mathieu a kol. Étude de l'intégration d'un jeu sérieux pour l'enseignement de la programmation dans différents contextes universitaires.* ResearchGate | Share and discover research [online]. September 2014 [cit. 2018-03-24]. Dostupné z: https://www.researchgate.net/publication/276411112

[35]. MURATET, Mathieu a kol. *Jeu sérieux et motivation des étudiants pour apprendre: influence du contexte avec Prog&Play*. ResearchGate | Share and discover research [online]. December 2012 [cit. 2018-03-24]. Dostupné z: https://www.researchgate.net/publication/276410844

[36]. O'GRADY, Michael J. *Practical Problem-Based Learning in Computing Education*. ACM Transactions on Computing Education [online]. 2012, 12(3), 1-16 [cit. 2018-03-24]. DOI: 10.1145/2275597.2275599. ISSN 19466226. Dostupné z: http://dl.acm.org/citation.cfm?doid=2275597.2275599

[37]. OLSSON, Marie a Peter MOZELIUS. *Game-based Learning and Game Construction as an E-learning Strategy in Programming Education.* GUIDE Association – Global Universities in Distance Education, 2016 [cit. 2018-03-24]. Dostupné z: https://www.researchgate.net/publication/304490353

[38]. PANAGIOTIS, Fotaris; THEODOROS, Mastoras; LEINFELLNER, Richard and Yasmine, ROSUNALLY. *Climbing Up the Leaderboard: An Empirical Study of Applying Gamification Techniques to a Computer Programming Class*. Electronic Journal of e-Learning, 14(2), pp. 94-110. ISSN 1479-439X [Article]. Dostupné z: https://www.researchgate.net/publication/293816223

[39]. PECINOVSKÝ, Rudolf, Jarmila PAVLÍČKOVÁ a Luboš PAVLÍČEK. *Let's modify the objects-first approach into design-patterns-first.* ACM SIGCSE Bulletin [online]. 2006, 38(3), 188- [cit. 2018-03-25]. DOI: 10.1145/1140123.1140175. ISSN 00978418. Dostupné z: http://portal.acm.org/citation.cfm?doid=1140123.1140175

[40]. PECINOVSKÝ, Rudolf. *Java 8: úvod do objektové architektury pro mírně pokročilé*. Praha: Grada Publishing, 2014. Knihovna programátora (Grada). ISBN 978-80-247-4638-8.

[41]. PECINOVSKÝ, Rudolf. *Současné trendy v metodice výuky programování*. Počítač ve škole 2006 [online]. Nové Město na Moravě: Gymnázium Vincence Makovského [cit. 2018-03-24]. Dostupné z: http://www.pocitacveskole.cz/system/files/uzivatel/9/clanky/pecinovsky_pdf_13597.pdf

[42]. PECINOVSKÝ, Rudolf. *OOP – learn object oriented thinking and programming*. Řepín: Tomáš Bruckner, 2013. Academic series. ISBN 978-80-904661-8-0.

[43]. PECINOVSKÝ, Rudolf. *PROČ A JAK UČIT OOP ŽÁKY ZÁKLADNÍCH A STŘEDNÍCH ŠKOL*. Žilinská didaktická konferencia, 2004 [cit. 2018-03-24]. Dostupné z: https://www.researchgate.net/publication/267694040

[44]. RAGONIS, Noa a Mordechai BEN-ARI. *On understanding the statics and dynamics of object-oriented programs.* ACM SIGCSE Bulletin [online]. 2005, 37(1), 226- [cit. 2018-03-24]. DOI: 10.1145/1047124.1047425. ISSN 00978418. Dostupné z: http://portal.acm.org/citation.cfm?doid=1047124.1047425

[45]. ROMO, Enrique Kato. *Game design for a serious game to help learn programming.* FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO [online]. July 2011 [cit. 2018-03-24]. Dostupné z: https://repositorio-aberto.up.pt/handle/10216/61659

[46]. SANDERS, Kate, Jonas BOUSTEDT, Anna ECKERDAL, Robert MCCARTNEY, Jan Erik MOSTRÖM, Lynda THOMAS a Carol ZANDER. *Student understanding of object-oriented programming as expressed in concept maps.* ACM SIGCSE Bulletin [online]. 2008, 40(1), 332- [cit. 2018-03-24]. DOI: 10.1145/1352322.1352251. ISSN 00978418. Dostupné z: http://portal.acm.org/citation.cfm?doid=1352322.1352251

[47]. SELLS, Chris. a Ian GRIFFITHS. *Programming WPF*. 2nd ed. Sebastopol, Calif.: O'Reilly, 2007, 747-749. ISBN 9780596510374.

[48]. SERRANO-LAGUNA, Angel, Javier TORRENTE, Borja MANERO a Baltasar FERNANDEZ-MANJON. *A game engine to learn computer science languages*. In: 2014 IEEE Frontiers in Education Conference (FIE) Proceedings [online]. IEEE, 2014, 2014, s. 1-8 [cit. 2018-03-24]. DOI: 10.1109/FIE.2014.7044112. ISBN 978-1-4799-3922-0. Dostupné z: http://ieeexplore.ieee.org/document/7044112/

[49]. SCHMIDT, Douglas C., Frank. BUSCHMANN a Kevlin. HENNEY. *Pattern-oriented software architecture*. New York: Wiley, 2007. ISBN 0-471-60695-2.

[50]. SCHMOLITZKY, Axel. *Teaching inheritance concepts with Java*. In: Proceedings of the 4th international symposium on Principles and practice of programming in

Java – PPPJ '06 [online]. New York, New York, USA: ACM Press, 2006, 2006, s. 203- [cit. 2018-03-24]. DOI: 10.1145/1168054.1168084. ISBN 3939352055. Dostupné z: http://portal.acm.org/citation.cfm?doid=1168054.1168084

[51]. STALLINGS, William. *Operating systems: internals and design principles*. 4th ed. Upper Saddle River, N.J: Prentice Hall, 2000. ISBN 0130319996.

[52]. SWACHA, Jakub. *Scripting Environments of Gamified Learning Management Systems for Programming Education*. ALEXANDRE PEIXOTO DE QUEIRÓS, Ricardo a Mário Teixeira PINTO, ed. Gamification-Based E-Learning Strategies for Computer Programming Education [online]. IGI Global, 2017, s. 278-294 [cit. 2018-03-24]. Advances in Game-Based Learning. DOI: 10.4018/978-1-5225-1034-5.ch013. ISBN 9781522510345. Dostupné z: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-5225-1034-5.ch013

[53]. TAHIR, Rabail a Alf INGE WANG. *State of the art in Game Based Learning: Dimensions for Evaluating Educational Games*. ResearchGate | Share and discover research [online]. December 2017 [cit. 2018-03-24]. Dostupné z: https://www.researchgate.net/publication/321654583

[54]. TENENBERG, Josh a Robert MCCARTNEY. *Editorial: Computational Tools for Computing Education*. ACM Transactions on Computing Education [online]. 2011, 11(4), 1-4 [cit. 2018-03-24]. DOI: 10.1145/2048931.2048932. ISSN 19466226. Dostupné z: http://dl.acm.org/citation.cfm?doid=2048931.2048932

[55]. VAHLDICK, A. and Antonio Jose MENDES and MARCELINO, M.J.P.; *Analysing the Enjoyment of a Serious Game for Programming Learning With two Unrelated Higher Education Audiences*, in European Conference on Games Based Learning, 2015 [publication]. Dostupné z: https://www.researchgate.net/publication/282666632

[56]. WEISS, Stephen. *Teaching design patterns by stealth*. ACM SIGCSE Bulletin [online]. 2005, 37(1), 492- [cit. 2018-03-24]. DOI: 10.1145/1047124.1047500. ISSN 00978418. Dostupné z: http://portal.acm.org/citation.cfm?doid=1047124.1047500

[57]. WICK, Michael R. *Teaching design patterns in CS1*. ACM SIGCSE Bulletin [online]. 2005, 37(1), 487- [cit. 2018-03-24]. DOI: 10.1145/1047124.1047499. ISSN 00978418. Dostupné z: http://portal.acm.org/citation.cfm?doid=1047124.1047499

[58]. *Design Patterns* | Object Oriented Design [online]. [Cit. 2018-03-24] Dostupné z: http://www.oodesign.com/