

Testování v herním průmyslu

Bc. Zdeněk Pohludka

Diplomová práce
2025



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav informatiky a umělé inteligence

Akademický rok: 2024/2025

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	Bc. Zdeněk Pohludka
Osobní číslo:	A22450
Studijní program:	N0613A140022 Informační technologie
Specializace:	Softwarové inženýrství
Forma studia:	Kombinovaná
Téma práce:	Testování v herním průmyslu
Téma práce anglicky:	Testing in Game Industry

Zásady pro vypracování

- Vypracujte literární rešerši na zadané téma.
- Analýzujte jednotlivé metody testování používané ve vývoji počítačových her.
- Navrhněte testovací scénáře a způsoby testování pro vybranou hru ve vývoji.
- Realizujte testování s využitím vhodných technik.
- Analýzujte a popište výsledky testování.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. PATTON, Ron. Testování softwaru. Praha: Computer Press, 2002. ISBN 80-7226-636-5.
2. ISTQB. Certified Tester Game Testing (CT-GaMe) Syllabus [online]. Brusel: ISTQB not-for-profit association, 2022 [cit. 2022-11-06]. Dostupné z: https://istqb-main-web-prod.s3.amazonaws.com/media/documents/ISTQB_CT_GaMe_Syllabus_v1.0.1_LtrKuyi.pdf.
3. Aleem, S., Capretz, L.F. & Ahmed, F. Game development software engineering process life cycle: a systematic review. J Softw Eng Res Dev 4, 6 (2016). <https://doi.org/10.1186/s40411-016-0032-7>.
4. Game Testing: All in One. 3rd ed. Dulles: Mercury Learning & Information, 2016. ISBN 9781942270768.
5. RAMADAN, Rido a Yani WIDYANI. Game development life cycle guidelines. Indonesia: IEEE, 2013. ISBN 978-1-4799-4692-1.

Vedoucí diplomové práce: **Ing. Tomáš Vogeltanz, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **27. října 2024**
Termín odevzdání diplomové práce: **2. června 2025**

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan



prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 17. listopadu 2024

PROHLÁŠENÍ AUTORA DIPLOMOVÉ PRÁCE

Beru na vědomí, že

- odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení diplomové práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – diplomovou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na diplomové práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Tato práce se zabývá komplexním přístupem k testování softwaru se zaměřením na herní průmysl a hru Preserve. V teoretické části jsou popsány základní pojmy, životní cyklus testování SW a rozdělení testů na funkcionální a nefunkcionální včetně specifik pro Unity Engine. Praktická část konkretizuje tvorbu testovacích scénářů, provedení funkcionálního, lokalizačního, kompatibilního, výkonnostního, compliance a automatizovaného testování, včetně sběru zpětné vazby a logování chyb. Výsledky ukazují silné a slabé stránky testované hry a na jejich základě jsou navržena doporučení pro optimalizaci testovacích postupů, využití AI a zefektivnění spolupráce s hráči. Práce přináší ucelený pohled na testování herního softwaru a návrhy pro budoucí projekty.

Klíčová slova: testování, herní průmysl, testovací techniky, videohra, Steam, Preserve, Unity

ABSTRACT

This thesis explores a comprehensive approach to software testing with a focus on the gaming industry and the game Preserve. The theoretical part describes the basic concepts, the software testing lifecycle and the division of tests into functional and non-functional, including specifics for the Unity Engine. The practical part specifies the creation of test scenarios, execution of functional, localization, compatibility, performance, compliance and automated testing, including feedback collection and bug logging. The results show the strengths and weaknesses of the game under test, and based on these results, recommendations are proposed for optimizing testing procedures, using AI, and streamlining collaboration with players. The paper provides a comprehensive view of game software testing and suggestions for future projects.

Keywords: testing, game industry, testing techniques, videogame, Steam, Preserve, Unity

Tímto bych rád nejprve poděkoval své ženě Anně za trpělivost při hlídání našeho syna a za neocenitelnou oporu během mého studia. Dále děkuji produkční a vydavatelské společnosti Grindstone a hernímu studiu Bitmap Galaxy za možnost být součástí týmu a podílet se na vývoji hry Preserve, bez níž by tato práce nemohla vzniknout.

Děkuji svému vedoucímu práce, Ing. Tomáši Vogeltanzovi, Ph.D., za cennou zpětnou vazbu a odbornou pomoc při psaní této práce. Děkuji Dominiku Mačenkovi a hernímu studiu NOXGames za poskytnutí informací o jejich testovacích procesech. Dále děkuji Lukáši Petrovi za pomoc při komunikaci se společností SCS Software s.r.o., Kristýně Routnerové a týmu SCS Software za sdílení detailů o testování.

Závěrem děkuji Lukáši Niklovi za zapůjčení hardwaru k testům a komunitě Games Development Community CZ/SK za cenné rady při tvorbě a realizaci testů.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Prohlašuji, že při tvorbě této práce jsem použil/a nástroj generativního modelu AI ChatGPT s adresou: <https://chatgpt.com> za účelem korektury textu, optimalizace kódu a brainstormingu. Po použití tohoto nástroje jsem provedl/a kontrolu obsahu a přebírám za něj plnou zodpovědnost.

OBSAH

ÚVOD.....	13
TEORETICKÁ ČÁST.....	14
1 SOFTWAREVÉ TESTOVÁNÍ.....	15
1.1 DEFINICE CHYBY.....	15
1.1.1 Error – Lidská chyba.....	16
1.1.2 Defect – Defekt.....	16
1.1.3 Failure – vada systému.....	17
1.2 ŽIVOTNÍ CYKLUS TESTOVÁNÍ SOFTWARE.....	18
1.2.1 Identifikace a analýza potřeb.....	18
1.2.2 Vytvoření testovací strategie.....	19
1.2.3 Návrh a vytvoření testovacího scénáře + příprava prostředí.....	19
1.2.4 Otestování SW.....	20
1.2.5 Nalezení a nahlášení chyb + následná oprava.....	20
1.2.6 Vytvoření reportu a uzavření testování.....	20
1.3 TYPY TESTOVÁNÍ.....	21
1.3.1 Manuální a automatizované testování.....	22
1.3.2 Testování černé, bílé a šedé skříňky.....	23
1.3.3 Testování v Unity Engine.....	23
1.4 FUNKCIONÁLNÍ TESTOVÁNÍ.....	25
1.4.1 Unit testování (Unit testing).....	25
1.4.2 Integrační testování (Integration testing).....	26
1.4.3 E2E testování (End-to-End testing).....	27
1.4.4 Lokalizační testování (Localization testing).....	28
1.4.5 Regresní testování (Regression testing).....	29
1.5 NEFUNKCIONÁLNÍ TESTOVÁNÍ.....	30
1.5.1 Testování dodržování předpisů (Compliance testing).....	30
1.5.2 Bezpečnostní testování (Security testing).....	32
1.5.3 Testování použitelnosti (Usability testing).....	33
1.5.4 Testování kompatibility (Compatibility testing).....	34
1.5.5 Výkonnostní testování (Performance testing).....	35
2 HERNÍ PRŮMYSL.....	37
2.1 VYVÍJENÉ PLATFORMY.....	37
2.1.1 PC.....	37
2.1.2 Konzole.....	38
2.1.3 Cloud.....	38
2.1.4 VR/AR.....	39
2.1.5 Mobilní platformy.....	39
2.2 PROFESE.....	40
2.3 ŽIVOTNÍ CYKLUS.....	42
2.3.1 Vodopádový model (Waterfall model).....	42
2.3.2 Scrum.....	43
2.3.3 Spirálový model (Spiral model).....	44
2.3.4 Lineární a iterativní modely navržené speciálně pro herní průmysl.....	45
2.4 TESTOVÁNÍ TECHNIKY.....	47
2.4.1 Kombinatorické testování.....	47
2.4.2 Testovací vývojové diagramy.....	48

2.4.3	<i>Cleanroom testování</i>	50
2.4.4	<i>Testovací stromy</i>	50
2.4.5	<i>Ad Hoc a Play testování</i>	52
PRAKTICKÁ ČÁST		53
3	TESTOVANÁ HRA PRESERVE	54
3.1	HERNÍ LOGIKA	55
3.1.1	<i>Herní UI</i>	55
3.1.2	<i>Bodovací systém</i>	56
4	TESTOVACÍ SCÉNÁŘE	59
4.1	TUTORIÁL	60
4.2	OVLÁDÁNÍ	63
5	TESTOVÁNÍ APLIKACE	64
5.1	ZAPISOVÁNÍ NALEZENÝCH CHYB	64
5.2	FUNKCIONÁLNÍ TESTOVÁNÍ (FUNCTIONAL TESTING)	66
5.2.1	<i>Biomy – chování základních prvků</i>	67
5.2.2	<i>Tutorial</i>	67
5.2.3	<i>Herní módy</i>	67
5.2.4	<i>Herní bodování</i>	67
5.2.5	<i>Herní prostředí</i>	68
5.2.6	<i>Menu, nastavení a UI</i>	69
5.2.7	<i>Puzzle</i>	72
5.2.8	<i>Ovládání</i>	73
5.3	LOKALIZAČNÍ TESTOVÁNÍ (LOCALISATION TESTING)	75
5.3.1	<i>Výstup testování</i>	77
5.4	TESTOVÁNÍ KOMPATIBILITY (COMPATIBILITY TESTING)	81
5.4.1	<i>Windows</i>	82
5.4.2	<i>Mac</i>	85
5.4.3	<i>Steam/Linux</i>	87
5.5	VÝKONNOSTNÍ TESTOVÁNÍ (PERFORMANCE TESTING)	90
5.5.1	<i>Testování spotřeby baterie a vykreslovacího výkonu (Battery Consumption & Rendering Performance Testing)</i>	90
5.5.2	<i>Testování vytrvalosti (Endurance testing)</i>	94
5.5.3	<i>Testování přerušení (Interrupt testing)</i>	95
5.6	TESTOVÁNÍ DODRŽOVÁNÍ PŘEDPISŮ (COMPLIANCE TESTING)	96
5.7	AUTOMATIZOVANÉ TESTOVÁNÍ	98
5.7.1	<i>Unit testování</i>	98
5.7.2	<i>Testování konfiguračních souborů</i>	101
5.7.3	<i>Spouštění testů</i>	104
5.8	TESTOVÁNÍ NA ZÁKLADĚ ZPĚTNÉ VAZBY (FEEDBACK TESTING)	105
6	PRAXE V JINÝCH STUDIÍCH	107
6.1	NOXGAMES	107
6.1.1	<i>Dotazník</i>	107
6.1.2	<i>Souhrn</i>	109
6.2	SCS SOFTWARE S.R.O.	109
6.2.1	<i>Dotazník</i>	109
6.2.2	<i>Souhrn</i>	111

7	ZHODNOCENÍ TESTOVÁNÍ A DOPORUČENÍ.....	112
7.1	SHRNUTÍ PRŮBĚHU TESTOVÁNÍ A HODNOCENÍ TESTOVACÍCH TECHNIK	112
7.1.1	<i>Funkcionální testování.....</i>	<i>112</i>
7.1.2	<i>Lokalizační testování</i>	<i>113</i>
7.1.3	<i>Testování kompatibility.....</i>	<i>114</i>
7.1.4	<i>Výkonnostní testování</i>	<i>115</i>
7.1.5	<i>Testování dodržování předpisů</i>	<i>116</i>
7.1.6	<i>Unit testování.....</i>	<i>117</i>
7.1.7	<i>Testování na základě zpětné vazby</i>	<i>118</i>
7.2	ANALÝZA VÝSLEDKŮ TESTOVÁNÍ	119
7.2.1	<i>Kvantitativní výsledky</i>	<i>119</i>
7.2.2	<i>Kvalitativní výsledky</i>	<i>123</i>
7.3	DOPORUČENÍ	125
7.3.1	<i>Rozdělit projekt na fáze.....</i>	<i>125</i>
7.3.2	<i>Budovat komunitu a „externí testery“</i>	<i>127</i>
7.3.3	<i>Zaměřit přiměřený čas do automatizovaného testování</i>	<i>127</i>
7.3.4	<i>Využití umělé inteligence</i>	<i>128</i>
	ZÁVĚR	129
	SEZNAM POUŽITÉ LITERATURY.....	131
	SEZNAM OBRÁZKŮ	138
	SEZNAM TABULEK.....	140
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	141
	SEZNAM PŘÍLOH.....	143
	PŘÍLOHA P I: SLOŽKA S PRACÍ.....	144

ÚVOD

Herní průmysl se stal jedním z největších odvětví současnosti a denně vznikají desítky nových her od velkých herních studií až po malé týmy či jednotlivce. Přestože doba postoupila a moderní technologie s novými postupy pomáhají vytvářet hry buďto rychleji nebo komplexněji než dříve, tak jejich kvalita v den vydání nebývá tak vysoká, jak se očekává a často přichází v prvních dnech aktualizace, které opravují chyby, na které tým buď nepřišel nebo je neměl čas opravit.

Cílem práce je sepsat nepoužívanější testovací techniky, které se v herním průmyslu používají a analyzovat jejich využití na testované hře Preserve. Ta byla vyvinuta menším slovenským týmem, který chtěl vytvořit relaxační logickou hru a vydal ji v rámci předběžného přístupu. Pro hru budou navrženy testovací scénáře a další způsoby testování, které mají zaručit co nejvíce optimalizovaný zážitek pro hráče a neztratit pozitivní hodnocení.

V rámci teoretické části budou vysvětleny definice chyby, životního cyklus testování softwaru a samotného typu testování. To kromě rozdělení na manuální, automatizované a podle typu skříňky nabídne i popis testování pomocí Unity Enginu a sepsání testovacích technik funkcionálního a nefunkcionálního testování. Následně se práce zaměří na specifika herního průmyslu a rozebere jednotlivé platformy, na které je možné hry vydat a nejčastější profese, které toto odvětví zaměstnává. Dále se soustředí na životní cykly, které se nejčastěji používají v herním průmyslu a specifické testovací techniky jako testovací stromy nebo kombinatorické testování.

Praktická část nabídne představení samotné hry Preserve, tvorbu testovacích scénářů, testování aplikace rozdělené podle metodik, zhodnocení výsledků testování, návrhy pro vylepšení a menší pohled na testování ve dvou jiných herních studiích.

V rámci analýzy budou vyhodnoceny výsledné testovací techniky. Byl popsán jejich postup testování, čas strávený testem, použité nástroje a souhrn pozitiv a negativ. Určení kvantitativních a kvalitativních výsledků práce na hře a sepsání seznamu doporučení, kde je navrženo například využití komunity nebo umělé inteligence.

Práce může být přínosná jak pro menší týmy, které hledají efektivní způsoby testování bez vysokého rozpočtu, tak pro testery, kteří chtějí vytvořit nebo upravit jejich testovací strategii potřebám herního vývoje. V neposlední řadě lze práci využít i jako výukový materiál pro vývojáře nebo testery.

I. TEORETICKÁ ČÁST

1 SOFTWAREVÉ TESTOVÁNÍ

Testování je proces kontroly kvality, který zahrnuje ověření splnění technických, designových a uživatelských požadavků. Tímto procesem se kontroluje, že software (SW) neobsahuje chyby a dodává záruku robustnosti. Testeři provádějí širokou škálu testů založených na různých scénářích, prostředích nebo uživatelském chování, které mohou způsobit chybné fungování aplikace. [1][2]

Testování je nezbytné z několika důvodů. Klíčovým přínosem je včasná detekce chyb, která minimalizuje jejich dopad a zajišťuje spolehlivější a včas dodaný produkt. Regresní testování pak udržuje či zvyšuje kvalitu samotného softwaru. Důvěru uživatelů navíc posiluje stabilní a bezchybný provoz aplikace. Nezanedbatelná je také brzká identifikace zranitelností a jejich následné zmírnění. [1][2]

Rolí v testování softwaru je více. Mezi nejdůležitější lze považovat Manažera testování (někdy nazývaný jako Lead tester), který má zodpovědnost ve všech akcích spjatých s testováním (plánuje testování, sestavuje týmy, vyhodnocuje úspěšnost týmu a jeho práce apod.) a bývá prostředníkem mezi vývojem a analýzou. Další role se zabývá přijetím vstupů od analytiků, jejich zpracováním a vytvořením výstupních dokumentů určených pro testování. Jedná se o roli Test analytika. Poslední (i nejčastější) rolí je Tester. Ten provádí samotné testování aplikace, zaznamenává objevené chyby a ty reportuje. [3]

1.1 Definice chyby

Kapitola se zaměřila na rozdíly mezi *lidskou chybou (Error)*, *chybou v softwaru (Defect)* a *projevením vady (Failure)* na softwaru. V češtině lze tato slova snadno zaměnit, ale ač jejich názvy mohou klamat, nelze je používat zaměnitelně.

Lidské chyby, jako časový tlak, složitost úkolu nebo únava, mohou zapříčinit vznik defektů (chyb nebo bugů), které mohou způsobit selhání systému. To vyobrazuje Obrázek 1. Defektem se rozumí odchylka mezi očekávaným a reálným výsledkem. Mohou se objevit např. v dokumentaci nebo zdrojovém kódu a pokud nejsou zavčas odhaleny, často způsobí vady nebo defekty v dalších fázích vývoje. Vady v systému mohou vzniknout jak z příčin uvedených výše, tak i vnějšími faktory jako například radiací, ohněm nebo vodou. Pro nalezení základní příčiny problému se musí objevit tzv. Kořenová příčina. [4][5][6]



Obrázek 1 Vzájemný vztah mezi lidskou chybou, defektem a vadou systému

(zdroj: vlastní)

1.1.1 Error – Lidská chyba

Lidská chyba může vzniknout v jakékoliv části vývoje. Ať už se jedná o prvotní analýzu s klientem anebo závěrečné testování před předáním klientovi. Tyto chyby lze kategorizovat na: [4][5][6]

- *chyby uživatelského rozhraní,*
- *syntaktické chyby,*
chyby ve výpočtu,
- *chyby řízení toku programu,*
- *chyby testování.*

1.1.2 Defect – Defekt

Jedná se o typ chyby, kterou tester odhalí buď při přípravě testů na základě dokumentace aplikace, při jejich provádění, nebo při vyhodnocování výsledků porovnáním se stanoveným očekávaným chováním. Pokud nejsou zavčas odhaleny, mohou způsobit vyšší náklady na opravu nebo problémy s celým systémem. Pokud se v kódu aktivuje vada, systém buď neprovede požadovanou akci, nebo naopak provede něco nežádoucího, což může vyústit v jeho selhání. Defekty se rozdělují podle tří kategorií. Rozdělení podle priority (*minimální priorita, střední priorita, maximální priorita*), dále podle závažnosti (*triviální závažnost, minimální závažnost, průměrná závažnost, kritická závažnost*) a podle pravděpodobnosti (*minimální pravděpodobnost, střední pravděpodobnost, maximální pravděpodobnost*). Kromě tohoto rozdělení je ale možné dělit i samotné typy defektů. [5][6][7]

- Jako první je chyba *Funkce*. Ta ovlivňuje schopnost hry, nebo to, jak hráč hru používá. V kódu nastane chyba, kdy buď daná funkce úplně chybí, nebo je nesprávně implementována.

- Chyba *Přiřazení* bývá způsobena, pokud je nesprávně nastavena hodnota, nebo chybí přiřazení dané hodnoty. Často se takto děje například při spuštění nového režimu hry nebo při začátku hry.
- Chyba *Kontroly* je charakteristická chybným ověřením dat. Jedná se konkrétně o chybějící podmínku nebo její chybnou definici. Pro jazyk C to může být například hodnota (*pointeru) porovnáváná s NULL místo adresy (pointer) – buď přímo z uložené proměnné, nebo jako vrácená hodnota z volání funkce.
- Další chybou je *Načasování*. Procesy závislé na času mohou mít problémy se sdílením zdrojů v reálném čase v případě, že jsou špatně načasovány. Uživatelsky přívětivý proces je vložení cut-scény nebo úvodní obrazovky s animací načítání. Tímto je ukázáno, že hra potřebuje čas pro zapsání/nahrání dat a není bráněno procesům s tím spjatým.
- Chyby *Buildu* vznikají v důsledku chyb při používání systému knihovny zdrojového kódu, při kontrole změn v souborech aplikace nebo při určování a kontrole toho, která verze se má sestavit.
- Chybou *Algoritmu* se rozumí výpočet nebo rozhodovací proces, který vrátí jiný výsledek, než bylo žádané. Může se jednat o algoritmy obstarávající dialogy postav, výpočet poškození postavy nebo třeba pravidla pro různá vylepšení.
- Jako vady *Dokumentace* se nejčastěji jeví chyby ve fixních souborech hry. Chyba ale není způsobena kódem, nýbrž v samotném zápisu souboru, který je poté volán a využíván aplikací. Mohou zde patřit grafické, zvukové, nebo textové soubory.
- Poslední chybou je chyba *Interface*. Nastává tehdy, když je problém s přenosem nebo výměnou informací. V kódu se chyby rozhraní objevují, když je něco špatně ve způsobu, jakým jeden modul volá druhý. Může se jednat například o chybějící parametr ve volání funkce, volání funkce s přeházenými parametry apod.

1.1.3 Failure – vada systému

Pokud se po vytvoření nebo provedení změn v kódu systém nechová podle představ, jedná se o vadu systému způsobenou defektem. Existují ale vady, kterých si uživatel nikdy nevšimne, jelikož postup pro jejich vytvoření je nepravděpodobný. K selhání může dojít v důsledku lidské chyby, úmyslného zásahu jednotlivce do systému nebo vnějších vlivů. Například přehřátím, namočením nebo jiným poškozením hardwaru (HW). Pokud software

není kompatibilní s hardwarem, pak také systém funguje neočekávaně. Ovšem pokud vzniknou chyby v důsledku nesprávného zacházení s testovacími daty, testovacím prostředím atd., tak tyto stavy se označují jako defekty, ačkoli se ve skutečnosti jedná o vadu systému. [4][5][6]

1.2 Životní cyklus testování softwaru

Životní cyklus testování softwaru (anglicky Software Testing Life Cycle – STLC) charakterizuje proces, během kterého se testuje SW za účelem zajištění kvality a stanovených požadavků. Každá fáze má svůj specifický účel, který musí být splněn, než se přejde do další fáze. V případě chyb spolupracují testéři s vývojáři či analytiky a součástí procesu může být i validace a verifikace produktu. [8][9]

S tímto procesem souvisí také životní cyklus vývoje softwaru (anglicky Software Development Life Cycle – SDLC), který se zaměřuje na fáze vývoje produktu. Je v něm zahrnuta jak počáteční analýza, tak i nasazení na produkci a údržbu. STLC je jeho podprocesem, který řeší kvalitu softwaru a jeho splnění požadavků. STLC začíná ve chvíli, kdy je zpřístupněn kód nebo požadavky. [8][9]

1.2.1 Identifikace a analýza potřeb

Cílem této fáze je porozumět chování produktu, odhalit slabiny a nesrovnalosti v dokumentaci a ověřit proveditelnost testování. Během analýzy se prochází funkcionální a nefunkcionální požadavky na software. Následně se podle toho určí, co se bude testovat manuálně a co automatizovaně, stanoví se priority testovacích požadavků a připraví se dokumentace. Dokumentace většinou obsahuje požadavky pro testování, kontrolu požadavků nebo zohlednění očekávání (např. pole pro email musí obsahovat symbol @). Klíčové kroky této fáze tedy jsou: [8][9]

- Identifikace modulů, které se mají otestovat.
- Určení toho, které testy provést.
- Definovat metody testování (manuální vs. automatizované).
- Sepsání požadavků (např. výběr prostředí nebo HW) a výstupů.

1.2.2 Vytvoření testovací strategie

Výstupem této fáze je dokument testovacího plánu, který obsahuje strategii, cíle a organizační rámec pro testování produktu. Dokument se vytváří na základě analýzy požadavků a definují se v něm prvky jako rozsah, cíle, harmonogram, potřebné nástroje a prostředí. Též se vyhodnocují rizika, rozděluje se zodpovědnost a odhaduje se časová a pracovní náročnost. Pro tuto fázi je tedy klíčové: [8][9]

- Vytvoření strategie, naplánování testů a sepsání test plánu.
- Při tvorbě plánu se zohlednit faktory jako rozpočet, čas, úsilí, prostředí a zdroje.
- Rozdělit se o povinnosti mezi členy týmu.

1.2.3 Návrh a vytvoření testovacího scénáře + příprava prostředí

Pro fázi návrhu a vytvoření testovacího scénáře (TS) se vytváří takové případy, které pokrývají co nejširším způsobem kombinace chování systému. Samotné testy mohou mít různou prioritu podle četnosti výskytu, dopadu na celkovou aplikaci, preferencím klienta apod. Vytváří se též automatizované skripty, validace požadavků, průběžná revize a schvalování testovacích případů. Hlavním výstupem jsou tedy testovací případy uspořádané do testovacích sad. [8][9]

Výstupem fáze přípravy prostředí je funkční testovací prostředí, které zahrnuje všechny klíčové konfigurace. Prostor pro testování musí obsahovat veškerý potřebný HW, SW, servery a nástroje pro hlášení chyb. Vytváří se tím simulace reálného prostředí s pravými uživateli a jejich chováním. Pro tuto fázi je důležité mít: [8][9]

- Vytvořené a definované testovací případy, které se zorganizují do testovacích sad.
- Napsané automatizované testovací případy.
- Vygenerovaná nebo jinak získaná potřebná data pro testování.
- Testovací sady, které mají jasnou prioritu a jsou systematicky spravované.
- Definice požadavků pro testovací prostředí a nastavení samotného prostředí.
- Provedení smoke testů k ověření funkčnosti prostředí.

1.2.4 Otestování SW

V této fázi testéři spouští testovací případy podle stanoveného testovacího plánu a porovnávají očekávané výsledky s těmi reálnými. Veškeré nesrovnalosti jsou zaznamenány a poslány k ověření nebo opravě. Opravené chyby se opět testují a je využíváno též regresního testování pro obecné ověření funkčnosti aplikace. Vzhledem k regresnímu testování je výhodné mít co nejvíce automatizovaných testů pro zkrácení času a vyšší efektivitu. [8][9]

1.2.5 Nalezení a nahlášení chyb + následná oprava

Cílem fáze je identifikace, zaznamenání a vyřešení chyb objevených při testování. Tester musí zaznamenat všechny neúspěšné testovací případy a buď je předat vývojáři k opravě, nebo ověřit očekávané chování s klientem či analytikem. Každá chyba by měla být důkladně zdokumentována (např. skrze nástroje jako Jira nebo Redmine). Chyba zahrnuje informace o skutečném a očekávaném výsledku, podrobný postup reprodukce chyby, doplňující materiály (video či obrázek) a organizační údaje, jako je závažnost, typ chyby a její priorita k opravě. Nástroje jako Jira se využívají i pro efektivní sledování životního cyklu chyb a jejich reportingu. Po dokončení opravy vývojář informuje testera, který ověří funkčnost opravy a provede případné regresní testy. Pro tuto fázi je důležité: [8][9]

- Důkladně popsat objevenou chybu do nástroje pro evidenci chyb.
- Přidat informace o prioritě, závažnosti a jejím typu.
- Po opravě provést otestování opravy chyby a regresní testování pro zajištění očekávaného chování.

1.2.6 Vytvoření reportu a uzavření testování

V poslední fázi nastává vyhodnocení celého procesu testování a předání finálního produktu. Tester hodnotí výsledky testů, míru pokrytí testy, dodržení harmonogramu, nákladovou efektivitu projektu a celkovou kvalitu produktu. V případě rozdílů od stanovených cílů se analyzují příčiny. V této fázi často dochází i k sebereflexi celého týmu a ke zhodnocení strategie, identifikaci problémů a zamyšlení nad možnými zlepšeními do budoucna. Pokud se jedná o pravidelné testování (např. skrze DevOps), tak se stanoví i frekvence a obsah reportů jednotlivých verzí. Fáze má tyto klíčové aspekty: [8][9]

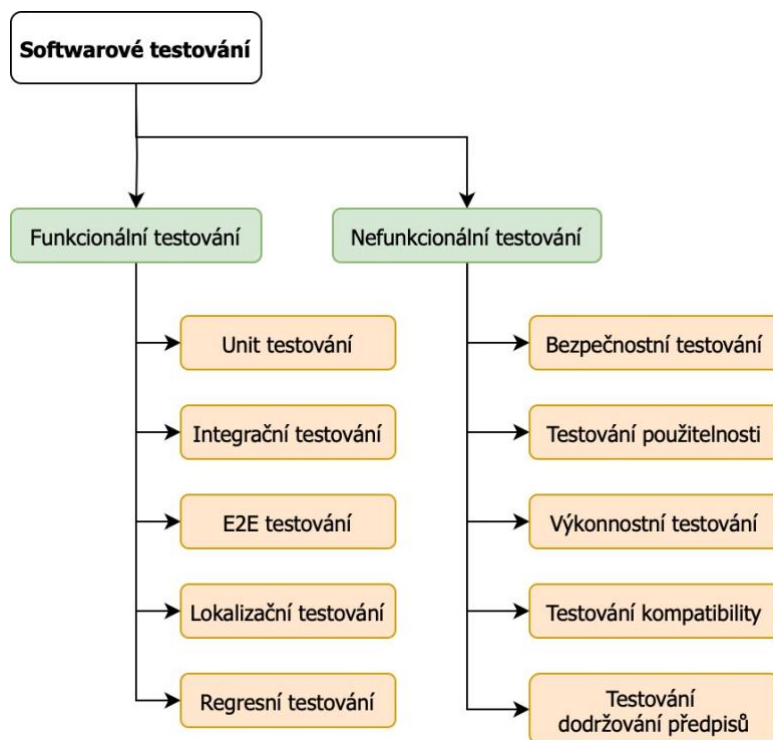
- Odevzdání finálního produktu.
- Sepsání závěrečné dokumentace a reportu o záznamech metrik z provedených testů.
- Sepsání zprávy o průběhu a výsledku testování.
- Zdokumentování všech odchylek od vytyčených cílů a jejich odůvodnění.
- Aplikování sebereflexe pro celý tým.

1.3 Typy testování

Pokud je požadované, aby aplikace fungovala podle stanovených parametrů a měla pozitivní dopad, musí projít testováním, které je často tříděno podle několika aspektů. I jednoduchá aplikace může vyžadovat širokou škálu různých testů. Testovací metody lze seskupit do několika základních kategorií. Obrázek 2 znázorňuje jednotlivé typy testování a jejich rozdělení do těchto skupin. [4][10]

První je manuální a automatizované testování, kdy se rozhoduje o tom, zdali test provádí člověk nebo stroj. Druhou skupinou je testování podle typu skříňky (černá, bílá nebo šedá). Určuje míru znalostí interních systémů během testování. Třetí základní skupinu tvoří funkcionální a nefunkcionální testování. Funkcionální testování ověřuje, zda aplikace pracuje podle předem stanovených požadavků, zatímco nefunkcionální testování hodnotí její spolehlivost, použitelnost a výkon. Tyto dvě skupiny se dále dělí na další metody, ty ale často mohou mít přesah napříč skupinami. Podrobněji je to rozebráno v další kapitole. [4][10]

Například Lokalizační testování primárně patří mezi funkcionální testování, ale jsou zde oblasti, které spadají do nefunkcionálního. Ověření správného překladu textů do různých jazyků patří k funkcionálnímu testování, zatímco kontrola, zda se rozvržení textu napříč jednotlivými překlady zachovává, spadá do nefunkcionálního testování. [11]



Obrázek 2 Typy testování

(zdroj: vlastní)

1.3.1 Manuální a automatizované testování

Mezi další dva přístupy k testování se považuje manuální a automatizované testování. Manuální testování je proces, při kterém testéři ručně ověřují funkčnost softwaru z pohledu koncového uživatele podle předem připravených testovacích scénářů a plánů. Je vhodné pro testování funkčnosti, uživatelského rozhraní, UX (user experience) a celkového chování aplikací, zejména u krátkodobých projektů nebo tam, kde je potřeba lidský úsudek. Mezi výhody patří nižší náklady, rychlá zpětná vazba a flexibilita. Nevýhodou je časová náročnost, nemožnost opakovaného použití testů a vyšší riziko lidské chyby. [1][12]

Automatizované testování využívá předem definované frameworky a skripty k provádění testů bez zásahu člověka. Je ideální pro opakující se úlohy, jako je regresní testování, a nabízí rychlé provedení, vyšší přesnost a široké pokrytí testovaných oblastí. Mezi jeho výhody patří úspora času a nákladů při dlouhodobém používání, eliminace lidských chyb a možnost opakovaného použití testů. Automatizace je vhodná tam, kde je potřeba škálovatelnost, efektivita nebo jsou omezené lidské zdroje. Mezi nevýhody patří vyšší počáteční náklady, potřeba zkušených odborníků a náročnější údržba testů při změnách softwaru. Jako případ nástrojů lze vzít NUnits, Cypress nebo Robot framework. [1][12]

Automatizované testování sice přináší rychlost, efektivitu a možnost opakovaného provádění testů, ale nemůže zcela nahradit manuální testování, zejména v oblastech vyžadujících lidskou intuici a kreativitu, jako je explorativní testování nebo testování použitelnosti. Firmy by však měly usilovat o postupný přechod k automatizaci, zejména kvůli rostoucím nárokům na rychlost vývoje a nasazování nových funkcí. Zatímco manuální testování je vhodné pro počáteční analýzu a menší rozsah, automatizace je klíčem k efektivnímu testování ve velkém měřítku a může rozhodovat o konkurenceschopnosti na trhu. [1][12]

1.3.2 Testování černé, bílé a šedé skříňky

Celkový postup při testování lze charakterizovat jako testováním černé, bílé nebo šedé skříňky. Rozdíl se nachází ve vědomostech testera, který provádí testování. Pokud tester zná pouze aplikaci na rovině jejího chování, ale nezná její podrobné interní procesy, jedná se o testování černé skříňky. Tester nemá přístup k vnitřní implementaci systému a může ověřovat pouze jeho výstupy. Pokud ale tester zná i interní procesy a má například přístup ke kódu, jedná se o testování bílé skříňky a tester má lepší predispozice k více optimálnímu testování. Na druhé straně může tento přístup vést ke ztrátě objektivity a přílišnému přizpůsobení testů konkrétnímu zdrojovému kódu a jeho omezením. Testování šedé skříňky je kombinací obou předchozích postupů a tester má povědomí o interních procesech, ale nemusí v nich být expert. [1][10]

1.3.3 Testování v Unity Engine

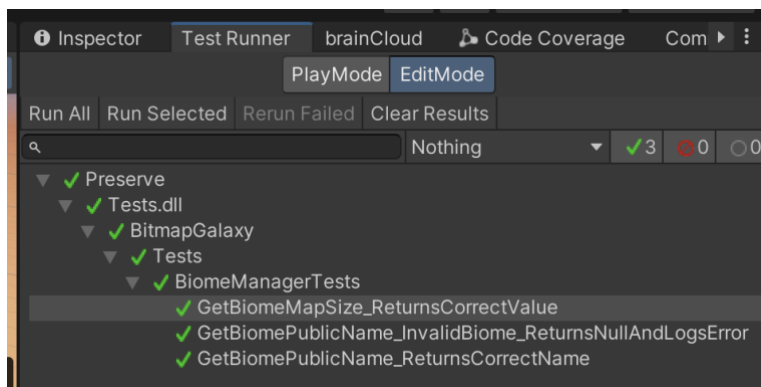
V Unity se doporučuje při psaní kódu používat Assembly definition (AD) soubory, které dovedou vytvářet z uceleného projektu menší sestavy, které se mohou spouštět separátně. Tento přístup se ukazuje jako efektivní zejména u středně velkých a rozsáhlých projektů, protože při sestavování (build) není nutné překládat celý zdrojový kód aplikace, ale postačuje kompilovat pouze moduly, v nichž došlo ke změnám, což současně podporuje modulární uspořádání kódu. [13][14]

Unity nabízí pro testování vlastní nástroj s názvem Unity Test Framework (UTF). Ten nabízí vývojářům framework pro automatizované testování v režimech Edit a Play Mode. Edit Mode testy běží v editoru a není při nich spuštěna hra. Jsou tedy rychlé a využívají se k testování logiky a funkcí. Play Mode testy zase testují UI (user interface), interakci a scény, jsou tedy podstatně pomalejší a při jejich spuštění je spuštěna i hra. Obecně to funguje tak, že UTF využívá AD soubory k prohledání zdrojového kódu hry, a testovací AD soubor na

nalezené komponenty odkazuje, čímž získává přístup k odpovídajícím testům. Testy samotné se spouští přes NUnits (Unit testovací framework pro .NET jazyky). Play a Edit testy musí být v rozdílných sestavách. Pro ovládání testů lze využít jak konzole, tak Test Runner okno (Obrázek 3), ve kterém je možné přepínat mezi Play a Edit testy a spouštět celou sadu nebo jen určité testy. UTF je vhodné pro Integrovaní a Unit testování. [13][14]

Kromě UTF je také často využíván balíček Code Coverage. Ten dokáže ukázat, které části kódu jsou a nejsou testovány a vytvářet HTML reporty, které ukazují statistiku pokrytí napříč celého projektu. Další používaný balíček je Unity Cloud Build. Ten přes cloud simuluje různé zařízení a dovede díky tomu hry lépe otestovat na určených sestavách. Zajistí tak konzistentní zážitek napříč platformami a identifikuje problémy specifické pro určitou platformu. [13][14]

Následuje Unity Profiler. Ten dovede měřit výkon samotné hry a je vhodný pro testování výkonnosti. Profiler má různé moduly zaměřující se na vykreslování, zvuk, paměť a síť. Může se připojit k síťovým zařízením nebo k perifériím připojeným k počítači a otestovat, jak hra běží na zamýšlené platformě pro vydání. Funguje díky instrumentaci, což znamená, že Profiler používá vlastní značky v kódu hry a měří s nimi podrobné informace o časování jako doba provedení kódu v každé značce. Značky jsou zabudovány přímo v Unity API, ale může si je nastavit i vývojář sám ke sledování konkrétních hodnot. Kromě Profileru vývojář může využít i balíček Performance Testing Extension, který umožní vytvářet vlastní měření pro výkonnostní testování a z nich lépe optimalizovat samotnou hru. [13][14][15]

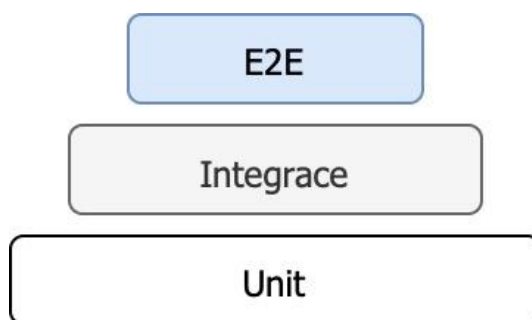


Obrázek 3 Test Runner okno v Unity

(zdroj: vlastní)

1.4 Funkcionální testování

Tento typ testování ověřuje, jestli aplikace funguje podle specifických požadavků a splňuje obchodní cíle. Zaměřuje se na testování manipulaci dat, interakci uživatelů, reakci systému, testování vstupu a výstupu. Většinou se testuje manuálně před nefunkčním testováním, kdy tester jen porovnává očekávané výsledky s těmi reálnými. Cílem je zajistit, aby všechny funkce fungovaly správně, a odhalit chyby či nedostatky ještě před uvedením produktu na trh. [16][17]



Obrázek 4 Testovací pyramida

(zdroj: vlastní)

Obrázek 4 charakterizuje tzv. Testovací pyramidu. Vyjadřuje myšlenku, že by se většina zdrojů měla soustředit na automatizaci unit testů a zároveň minimalizovat náklady na E2E (End-to-End) testování. Jako první odhad se doporučuje poměr: [18][19]

$$70 \text{ (Unit)} / 20 \text{ (Integrace)} / 10 \text{ (E2E)} \quad (1)$$

Odhad se později více přizpůsobí potřebám projektu, ale je nutné mít pořád tendenci trojúhelníku. Nedoporučuje se používat vzory, jako je převrácená pyramida nebo přesýpací hodiny. V modelu převrácené pyramidy se tým tenduje zaměřovat především na E2E testy, zatímco v modelu přesýpacích hodin klade důraz na velké množství E2E a unit testů, přičemž integrační testy jsou často opomíjeny a nedostatečně pokryty. [18][19]

1.4.1 Unit testování (Unit testing)

Unit testování dohlíží na testování nejmenších funkčních jednotek kódu (většinou metody nebo funkce). Obvykle tyto testy píší sami vývojáři už při psaní samotného kódu, čímž vytvářejí ověřovací testy, které zajišťují, že kód bude fungovat podle očekávání i v budoucnu. Hlavním účelem unit testování je tedy zajistit, aby jednotlivé části kódu pracovaly správně a splňovaly požadavky. To umožní rychlejší odhalení chyb při změnách kódu a podpoří efektivní vývoj. [20][21][22]

Pro pokrytí většiny testovacích případů je nutné využít klíčových aspektů. Testy musí ověřovat logiku systému a ověřovat hraniční hodnoty pro test i krajních a neplatných vstupů. Pokud tyto neplatné vstupy přijdou, musí být správně odbaveny. K tomu se využívá testování pro zpracování chyb. Vše záleží na rychlosti – testy by neměly zbytečně zdržovat vývoj a zároveň by měly být nezávislé na externích faktorech, jako jsou souborové systémy nebo databáze. Výsledky těchto testů se určují automaticky a nevyžadují lidský zásah. Nakonec samotné testy nesmí být složitější než testovaný kód a nesmí se s nimi přecházet do integračního testování. [20][21][22]

Mezi hlavní benefity unit testování patří včasné odhalení chyb, snadná automatizace, tvorba tzv. kódové dokumentace a lepší debugging. Jelikož sám vývojář většinou testy spouští po napsání nového kódu, nemusí čekat na ostatní a může rovnou případné chyby opravit. A v případě, že vývojář má např. napsat rozšiřující funkce nebo optimalizovat stávající kód, může použít unit testy jak pro zmapování chování, tak i případný debugging. Na druhou stranu jeho počáteční investice může být vysoká a pokud není vytvořen kvalitní testovací plán, může vzniknout buď nadměrné množství testovacích případů, které se těžko udržují aktualizované, nebo naopak nedostatečné pokrytí, které může způsobit falešný pocit spolehlivosti kódu. V obou případech nastává zbytečné vyhazování peněz a může být tedy považován jako zbytečný náklad. [20][21][22]

1.4.2 Integrační testování (Integration testing)

Po unit testování obvykle následuje integrační testování. Ačkoliv se tyto dvě metody často zaměňují, unit testování ověřuje nejmenší funkční jednotky kódu, zatímco integrační testování se zaměřuje na jejich vzájemné propojení a správnou spolupráci. Cílem je zajistit bezchybnou interakci jednotlivých komponent a spolehlivý přenos dat mezi nimi. Konflikty mohou nastat z těchto důvodů: nekonzistentní logika kódu, nedostatečné ošetření chyb nebo kolize při využívání služeb třetích stran. Je dobré mít vždy před testováním jistotu, že všechny moduly byly samostatně otestovány, aby se vyloučilo, že je problém na straně modulů a testy byly spolehlivé. Doporučuje se vytvořit test plán, kde je sepsána strategie, rozsah, cíle a testovací případy. Pokud je to možné, je výhodné co nejvíce procesů zautomatizovat a po každé změně provést regresní test, aby se ověřilo zachování funkčnosti. Integrační testování zastává čtyři přístupy, které jsou: [23][24]

- *Big-Bang testování* je přístup, ve kterém jsou všechny moduly spojeny a testovány najednou po dokončení unit testování. Pro svou rychlost je vhodné pro malé systémy, ale zahrnuje obtížnější odstranění chyb a ztíženou diagnostiku.
- *Inkrementální testování* funguje na principu rozdělení modulů na skupiny, které spolu úzce souvisí a postupně se testují další, dokud není vše otestováno. Vyžaduje důkladnější plánování, ale nabízí dřívější odhalení chyb a větší flexibilitu.
- *Bottom-up testování* se zaměřuje na testování od nejmenších modulů až po ty největší. Takto se postupně tester dostane od nejnižších úrovní až po nejvyšší a lze tak sledovat postupný vývoj. Využívá tzv. Drivers (ovladače) pro předání dat modulům nižší úrovně. Tento přístup je výhodný zejména v případech, kdy je komplexita na nižší úrovni.
- *Top-down testování* má opačný přístup než Bottom-up testování a zaměřuje se na systémy, kdy je větší komplexita v modulech vyšší úrovně. Nižší moduly ještě nemusí být integrovány, jsou tedy nahrazeny tzv. Stubs (pilíři). Nabízí snazší identifikaci chyb a možnost odděleného ladění modulů. Nevýhoda je ve vytváření stubs a menšímu testování nižších modulů.
- *Hybridní (Sandwich) testování* je kombinací Top-down a Bottom-up přístupů, čímž překonává jejich nedostatky. Testování probíhá simultánně a je vhodné pro velké projekty. Nevýhody spočívají ve vysoké ceně a nevhodnosti pro menší systémy, které mají vysokou vzájemnou závislost modulů.

Použitím vhodných přístupů lze dosáhnout téměř bezchybné integrace modulů, včas odhalit problémy dříve, než by se staly nákladnými, a zajistit stabilitu aplikace. Problém může nastat se samotným výběrem přístupu, který může být někdy komplikovaný. Též je potřeba myslet na časovou náročnost a složitost, která může vzniknout např. při používání pomocných komponent. A v neposlední řadě se může testování dost prodražit. [23][24]

1.4.3 E2E testování (End-to-End testing)

Tato technika testuje kompletní aplikaci z pohledu uživatele a zaměřuje se na všechny její části, jako jsou UI, databáze, API a servery. Testuje se kompletní systém od začátku do konce a pomáhá takto odhalit systémové chyby. Může se zaměřovat s akceptačním testováním, které je rozdílné v tom, že testy provádí klient, zatímco E2E testuje vývojářský tým. Cílem je zajistit, aby celý systém fungoval bez problémů a splňoval stanovené

požadavky. Jde nad rámec unit testování a integračního testování, protože posuzuje chování aplikace jako celku. [25][26][27]

Pro testování E2E je doporučeno držet se postupů, které zajistí jeho správné provedení. Mezi ně patří dostatečné pokrytí testovacích případů a jejich pravidelná aktualizace, prioritizace kritických workflow, vynechání testování výjimek a minimalizace testování UI, přičemž místo něj se doporučuje testovat logiku prostřednictvím API. Toto vyústí v měření úspěšnosti, které je důležitou metrikou pro analýzu výsledků. Zde lze využít pokrytí testů nebo hustotu defektů, které hodnotí míru pokrytí testů a sledování počtu chyb na jednotku kódu. Dále je dobré používat MTTD (Mean time to detect) a MTTR (Mean time to repair) pro zjištění doby nalezení chyby a následné opravy. Nejpoužívanější metody pro E2E testování jsou: [25][26][27]

- *Horizontální testování*, které má za cíl zajistit bezproblémovou interakci a funkčnost aplikace napříč různými komponentami. Testuje aplikaci na různých platformách, rozhraních nebo aplikacích, které navzájem spolupracují. Simuluje vnímání koncového klienta a kontroluje kompatibilitu mezi platformami, integrační testy a end-to-end procesy.
- *Vertikální testování*. Tento typ testování zajišťuje správnou interakci každé vrstvy nebo komponenty s vrstvami nad ní a pod ní. Cílem je především ujistit se, že správná data jsou přenášena na správné místo a ve správný čas, aby mohly být hladce prováděny všechny procesy na pozadí, které udržují aplikaci v chodu.

E2E zajišťuje komplexní přístup, který garantuje, že aplikace se chová podle reálných očekávání. Výhodami tedy může být průchod kompletní aplikace včetně všech komponent a testování podle testovacích scénářů založených na skutečném chování v aplikaci. Na druhou stranu testy mají dlouhou dobu trvání, mohou být náročné na údržbu a mohou vyžadovat větší kapitál pro provedení. [25][26][27]

1.4.4 Lokalizační testování (Localization testing)

Díky lokalizačnímu testování se ověřuje, zda software nebo aplikace splňuje náležitosti daného regionu a je přizpůsoben všem jazykovým, regionálním a kulturním normám. Cílem je zajistit, aby byl produkt správně přeložen, bral v úvahu formáty dat, měnu, slang termíny, místní zvyklosti a podobně, aby byl uživatelský zážitek co nejautentičtější. [28][29]

Mezi KPI (Key Performance Indicator) se dají zahrnout požadavky na softwarovou podporu nebo hodnocení spokojenosti s lokalizací a případné hlášení chyb. Mimo to se testy zaměřují na správnost překladu a kontrolu kulturní relevance. Správné formátování data, času nebo měny (např. japonské formátování data 2024年12月13日 vs. české formátování 13. 12. 2024). Testování uživatelského rozhraní ověřuje správné zobrazení a uspořádání textů i grafických prvků. Mezi testovací techniky patří: [28][29]

- *Testování použitelnosti a UI* může ověřit, zda aplikace je uživatelsky přívětivá a veškeré textace a grafické prvky jsou správně vloženy a intuitivní.
- *Testování překladů* zajistí, aby všechny textace byly správně přeloženy a byly dobře srozumitelné pro uživatele dané země/regionu.
- *Testování kulturní vhodnosti* kontroluje, zda design a obsah aplikace respektují zvyklosti cílové země a kulturní normy, aby se předešlo nedorozuměním a urážkám.

Toto testování zajišťuje kulturní a jazykovou připravenost aplikace na cílový trh, což automaticky vede k lepšímu přijetí u hráčů a větší pravděpodobnosti k úspěchu na daném trhu. Mezi nevýhody ale patří zvýšené náklady a časové nároky spojené s přizpůsobením (najmutí překladatelů, nelezení odborníků na danou kulturu, přizpůsobení UI pro jiný textový a grafický výstup atd.). [28][29]

1.4.5 Regresní testování (Regression testing)

Nelze jej přímo zařadit mezi funkcionální nebo nefunkcionální testování, protože jeho podstata je v ověření, že aplikace i po úpravách funguje stále správně a nevznikly negativní dopady. Cílem je odhalit problémy, které mohly vzniknout při zavádění nových funkcí nebo aktualizací či optimalizací funkcí stávajících. A přestože se jedná o důležitý aspekt testování, podle serveru Gartner investuje 54 % dotázaných do regresního testování pouze 1–25 % svého rozpočtu a 28 % věnuje 26–50 % rozpočtu [30]. Regresní testování se může plést s retestováním. Retestování se aplikuje při ověření neúspěšných testovacích případů, aby se ověřilo, že po opravě již funguje správně. Regresní testování má určit, zda nové změny nezpůsobily problémy se zaběhlými funkcemi. [31][32]

Tento typ testování je možné provést několika způsoby. Techniky se zaměřují na podmínky, jako je povaha změn nebo velikost dotčené oblasti. Techniky tedy jsou: [31][32]

- *Retestování všeho*, jak z názvu napovídá, je zaměřeno na kompletní test aplikace. Často bývá provedeno, pokud proběhlo více změn napříč systémem a je potřeba ověřit

celkové chování. Testování je velmi nákladné časově i zdroji, pokud se jedná o velkou aplikaci.

- *Výběr regresních testů* se soustředí na části, kde v poslední době proběhl větší počet úprav, nebo které jsou často používány. Zúžením na vybrané jednotky se snižují náklady a množství odvedené práce.
- *Prioritizace testovacích případů* se zaměřuje na základě vybraných parametrů, jako jsou například nejčastěji používané funkce, míra selhání funkcí a obchodní dopad určitých funkcí na testování určitých testovacích případů. Tím se opět zúží dotčená oblast a je možné tím snížit náklady na provedení.
- *Hybridní testování* kombinuje prioritizaci podle testovacích případů a výběr regresních případů do jednoho balíčku a lze díky tomu vybírat testovací případy podle potřeby. To může optimalizovat zdroje a čas na potřebné sekce a vynechat oblasti, které nejsou prioritní.

Mezi výhody tohoto testování patří zajištění kompatibility napříč prostředím, podpora automatizace a především ověření stability systému i po zavedení nových funkcí. Nevýhoda ale může tkvět v možné náročnosti na zdroje, komplexity testování a možného zpomalení vývoje, pokud není nastavena správná strategie pro testování. [31][32]

1.5 Nefunkcionální testování

Tento typ testování se zaměřuje na aspekty SW, jako jsou použitelnost, výkonnost, škálovatelnost, spolehlivost a pomáhá splnit očekávání koncového zákazníka. Často jeho výsledky nejsou ANO/NE, jak je tomu u funkcionálních testů, ale nabízí spíše měřitelné a stupňované odpovědi. Testování využívá automatizované nástroje a zpětnou vazbu reálných uživatelů k simulaci skutečného prostředí a analýze chování softwaru.[16][33]

1.5.1 Testování dodržování předpisů (Compliance testing)

Testování dodržování předpisů je proces ověřování, zda SW splňuje požadované regulace, specifikace a normy. Při testování se hodnotí tyto atributy: robustnost, funkce, výkonnost, interoperabilitu a chování systému. Pro správné otestování je dobré mít hlavní vývoj ukončen a aplikace by měla splňovat očekávání podle dokumentace. Integrační i funkční testy musí být úspěšně dokončeny a všechny licence aktivní. Cílem je zajistit, že software

odpovídá stanoveným standardům, je správně dokumentován a nevyvolává regulační problémy. [34][35]

Testování ověřuje, jestli aplikace splňuje všechny předpisy, normy a specifikace stanovené regulačními orgány. Také kontroluje soulad s právními požadavky, bezpečnostními standardy a předpisy, které jsou specifické pro určité odvětví. Tímto se minimalizuje riziko právních sankcí, zakázání aplikace v určitém regionu nebo na specifické platformě. Typy testování mohou být následující: [34][35]

- Jelikož v některých zemích je právně vyžadováno provést některé testy, lze zde zařadit *Povinné testování nařízené státem*. Například pro software, který se dotýká bezpečnosti, je povinné splnit normy, které testují přímo státní agentury nebo agentury jmenované státem. Pro splnění musí mít platnou certifikaci, jinak obdrží pokutu, musí uhradit náhradu škody, jinak bude produkt zakázán. Další formou může být *povinné testování nařízené jinou firmou*, které se uplatňuje mezi obchodními partnery, kdy jedna společnost může požadovat testovací report od druhé. [34][35]
- *Dobrovolné testování* provádí většinou společnosti třetích stran, které mají zaručit, aby proces ověření proběhl nestranně. Nejedná se zde o zákonnou povinnost, ale pouze o co nejlepší zavedení produktu. Podnik může také využít tzv. *interního testování*, které zapojí týmy interně, aby provedly testy shody s cílem zlepšit výkonnost svých produktů a služeb. [34][35]

Testy se také řídí řadou mezinárodních standardů, které stanovují organizace jako ISO (International Organization for Standardization), IEEE (Institute of Electrical and Electronics Engineers) nebo W3C (World Wide Web Consortium). Mezi tyto standardy se řadí ISO 9001 (řízení kvality), ISO/IEC 27001 (informační bezpečnost), GDPR (General Data Protection Regulation) nebo HIPAA (Health Insurance Portability and Accountability Act). [34][35]

Jako přínosy lze považovat zajištění souladu aplikace s normami a specifikacemi, které zvýší její důvěryhodnost. Ověří interoperabilitu, přenositelnost a dovede identifikovat oblasti, které je potřeba upravit a které jsou v rámci požadavků přijatelné. Naopak jako zápory patří vyšší náročnost na zdroje, jako jsou čas, finance a odborné znalosti norem a regulací. Někteří kritici varují na rigiditu, která omezuje inovace a vytváření falešného pocitu bezpečí, protože splnění regulací nezajistí odolnost aplikace proti všem hrozbám. [34][35]

1.5.2 Bezpečnostní testování (Security testing)

Cílem bezpečnostního testování je, aby aplikace byla chráněna před různými druhy bezpečnostních rizik, které mohou způsobit ztrátu dat nebo renomé společnosti. Odhaluje zranitelná místa v systému a zjišťuje, zda jsou data a prostředky systému chráněny před případnými narušiteli. Vyhodnocuje celkovou bezpečnost systému, identifikuje zranitelnosti a zajistí, aby byl systém chráněn proti neoprávněnému přístupu, narušení dat a dalším problémům souvisejícím se zabezpečením. [36][37][38]

Mezi hlavní aspekty hodnocení patří zabezpečení autentizace a autorizace (ochrana proti útokům typu brute force či session hijacking), komplexní ochrana dat v databázi i při přenosu, odolnost vůči útokům jako SQL Injection, XSS (Cross-Site Scripting) či CSRF (Cross-Site Request Forgery), vysoká dostupnost aplikace a ochrana proti neoprávněné manipulaci či crackování. Jako klíčové metriky lze poté využít počet nalezených zranitelností, míru úspěšnosti útoku, průměrnou dobu detekce a reakce na útok, anebo například počet úspěšných penetračních testů. [37][39]

Bezpečnostní testování má několik typů, které se zaměřují na různé druhy bezpečnostních rizik. Zde je pár příkladů: [36][37][38]

- *Penetrační testování* simuluje útok útočníka do systému. Etničtí hackeři se pokouší nabourat do systému a nalézt slabiny, na které se poté zaměří oprava.
- *Bezpečnostní skenování a skenování slabin* je většinou automatizovaný proces, aplikace se prochází skenem a naleznou se tím potenciální slabiny a chyby.
- *Testování sociálního inženýrství* simuluje chování při phishingu, baitingu apod. Tento test se zaměřuje spíše na lidskou stránku.
- *Posouzení stavu bezpečnosti* zahrnuje kompletní analýzu a přezkoumání zásad a postupů zabezpečení s cílem nalézt mezery.

Jedná se o složitý proces, který je náročný na odbornost a zdroje. Ve všech případech nemusí identifikovat veškeré hrozby a může přinést falešně pozitivní nebo negativní výsledky, které akorát povedou ke zmatkům a časové ztrátě. Navíc se některé útoky těžko simulují a je tedy těžké předvídat, jak budou útočníci se systémem pracovat. Přes tyto negativa ale může stále pomoci identifikovat dost zranitelných míst, aby to odradilo některé útočníky. Také celkově pomůže zabezpečit systém, sníží rizika spojené s bezpečnostními

incidenty a vybuduje pohotovější reakci na tyto incidenty. Pomůže též splnit bezpečnostní předpisy jako HIPAA nebo SOC2 (Service Organization Control 2). [36][37][38]

1.5.3 Testování použitelnosti (Usability testing)

Testování použitelnosti lze definovat jako metodu hodnocení uživatelské přívětivosti aplikace na základě testování s reálnými uživateli. Uživatelé plní různé úkoly a sleduje se jejich úspěšnost, postup a případné potíže. Tím se identifikují problematické oblasti a vytvoří se návrhy na zlepšení. Cílem není tedy jen efektivita ovládání, ale i celková spokojenost uživatelů s aplikací a její případná úprava pro lepší uživatelský zážitek. Mezi toto testování se nezahrnují dotazníky, uživatelské testování, A/B testování nebo focus skupiny. Ani jeden ze zmíněných testů nepředvede reálné chování uživatele v aplikaci při plnění specifických podmínek. [40][41][42]

Pro měření použitelnosti lze využít metriky jako procento dokončení úkolu, čas na obrazovce, čas strávený na úkolu, míra spokojenosti uživatele nebo míra chybných kliknutí. Také se může analyzovat optimální cesta z bodu A do bodu B a metrika úspěšnosti založená na počtu uživatelů, kterým se nepodařilo dostat do cíle. Výsledky těchto hodnot je dobré srovnat i s výsledky dřívějších verzí nebo konkurenčních produktů. [40][41][42]

Existuje několik typů testování použitelnosti, které se liší převážně informacemi, které jsou nutné získat, rozpočtem a volným časem na provedení testů. Mezi typy patří: [40][41][42]

- *Vzdálené vs. osobní testování* se odvíjí od místa konání testu. Vzdálené testování probíhá online při sdílení obrazovky a zaznamenáváním své aktivity. Toto testování bývá levnější, ale neposkytuje hlubší vhled do uživatelského chování. Osobní testování se odehrává fyzicky v laboratoři nebo jiné místnosti, kde výzkumník sleduje chování uživatele a analyzuje verbální i neverbální reakce. Toto testování bývá z pravidla dražší.
- *Kvalitativní nebo kvantitativní testování*. Kvalitativní se zaměřuje na subjektivní emoce, zkušenosti a problémy uživatelů. Díky metodám jako moderované testy zaznamenává uživatelské reakce a rozhovory, které přispívají k hlubší analýze. Kvantitativní testování se specializuje na shromáždění a analýzu datových údajů jako, jsou míra úspěšnosti, doba ukončení úkolu, hodnocení spokojenosti nebo míra chybovosti. Vytváří se různé vzorce a předpovědi pro zobecnění zjištění. V ideálním případě se kombinuje přístup obou typů testů.

- *Moderovaný vs. nemoderovaný přístup.* Moderované testování probíhá tak, že uživatel je vedený moderátorem, který ho může vést a odpovídat na otázky. Tento přístup je lepší při kvalitativním testování. Při nemoderovaném přístupu je uživatel sám a jeho aktivitu zaznamenává software pro sledování jeho činnosti a odpovědí. Činnosti účastníků nejsou kontrolovány ani řízeny.

Je to dobrý způsob, jak zjistit potřeby zákazníka, pro kterého je aplikace vytvořena. Tímto přístupem lze odhalit skryté problémy, které by interním testováním mohly zůstat neodhalené, a současně je to efektivní způsob, jak během vývoje ověřit, že tým pracuje správným směrem. Rozpoznáním potřeb může splnit očekávání, a proto ji budou uživatelé využívat častěji. Bohužel s tím jde i riziko, že výsledky mohou být diskutabilní, protože při výběru náhodných lidí se nikdy nestane, že se bude reprezentovat 100 % reálných zkušeností. S tím se pojí i výběr cílové skupiny, který může být velmi problematický. Např. pokud se vyvíjí aplikace pro děti ve věku 2–5 let, je těžké najít cílového uživatele. Tato metoda je však spojena s náklady na provedení, které často bývají vysoké. [40][41][42]

1.5.4 Testování kompatibility (Compatibility testing)

Testování kompatibility je typ testování softwaru, který zkoumá a porovnává funkčnost na různých prohlížečích, zařízeních, platformách a operačních systémech s důrazem odhalit potenciální nesrovnalosti. Cílem je zajistit, aby aplikace poskytovala konzistentní a bezproblémový uživatelský zážitek bez ohledu na použité prostředí. [43][44]

Jak již bylo zmíněno, testování ověřuje správné fungování aplikace na různých HW konfiguracích s různými OS a různých síťových prostředích. Mezi hlavní aspekty vyhodnocení patří pokrytí testů, které ukazuje počet nalezených chyb. Testování kompatibility se skládá z několika druhů. Mezi nejčastější patří: [43][44]

- *HW kompatibilita* kontroluje, že daná aplikace funguje správně na vybraných komponentech. K těmto testům může patřit i testování na různých zařízeních (mobil, počítač, konzole).
- *SW kompatibilita* je důležitá pro svou kontrolu chodu aplikace na jiném softwaru (např. Microsoft Word dokument lze otevřít v LibreOffice). Zároveň se ověřuje, že aplikace funguje na různých OS nebo s různými databázemi.
- *Backward compatibility testing* je technika pro ověření chování aplikace s jejími staršími verzemi nebo se starším HW.

- *Forward compatibility testing* ošetřuje, zda aplikace bude fungovat s novějšími verzemi nebo s novějším HW.

Mezi nevýhody tohoto testování patří vyšší náklady spojené s nákupem SW a HW v různých kombinacích, které jsou nejčastější pro uživatele (lze využít například Steam statistiky). Na druhou stranu toto testování zaručí spokojenost vyššího počtu uživatelů právě díky použití různých variant HW a SW. Též pomůže aplikaci více vyladit a zoptimalizovat. [43][44]

1.5.5 Výkonnostní testování (Performance testing)

Pro otestování škálovatelnosti, rychlosti a stability systému se používají výkonnostní testy. Ty mají jako hlavní cíl zajistit, aby systém splňoval výkonnostní požadavky koncových uživatelů. Pomohl identifikovat případné problémy, které by mohly způsobit pád systému nebo jeho zpomalení za určitých podmínek a poskytoval uživatelům pozitivní zkušenosti s aplikací. [45][46]

Testování výkonu je postup, při kterém se hodnotí, jak systém funguje z hlediska odezvy a stability při určitém pracovním zatížení. Mezi nejčastější problémy, které testy odhalí, jsou: [45][46]

- *Doba odezvy*, tedy čas mezi zadáním informací uživatelem do aplikace a reakcí aplikace na tuto akci, je delší, než by bylo žádoucí.
- *Nadměrná doba načítání* je delší, než doba nezbytně nutná ke spuštění aplikace.
- *Omezená škálovatelnost* nastává, když se aplikace špatně přizpůsobuje různému počtu uživatelů, například když aplikace s rostoucím počtem uživatelů začne mít problémy s výkonem.
- *Úzká místa (Bottlenecks)* jsou překážky v systému, které snižují celkový výkon aplikace. Obvykle jsou způsobena problémy s hardwarem nebo špatně napsaným kódem. *Využívání HW*, kde se například řadí vytížení RAM, CPU nebo GPU, může způsobit jak zpomalení hry, tak i samotného systému, na kterém hra běží.
- Procento požadavků, které vede k chybě, se nazývá *Míra chybovosti*. Slouží k identifikaci případných problémů, které mohou způsobovat chyby a zpomalení.

Výkonnostní testování obsahuje několik druhů, které se mohou vykonávat různě v době cyklu vývoje. Mezi nejpoužívanější patří *testování hrotů* (Spike testing), *stresové testování*

(Stress testing), *testování načítání* (Load testing), *testování vytrvalosti* (Endurance testing) a *testování přerušení* (Interrupt testing). [45][46]

- *Stresové testování* se provádí s účelem zátěže systému nad klasický rámec pracovních podmínek, čímž se zjistí jeho stabilita. Dokáže se tím určit bod zlomu a chování, které nastane po jeho překročení.
- *Testování hrotů* ověřuje měření výkonnosti systému při náhlých a nepředvídatelných skocích v používání. Na rozdíl od zátěžového testování bere testování hrotů v úvahu počet uživatelů a složitost prováděných činností (tedy nárůst několika generovaných obchodních procesů).
- *Testování načítání* zahrnuje testování schopnosti systému zvládnout velký počet uživatelů nebo transakcí současně. Umožňuje pochopit, zda je nutné upravit velikost architektury aplikace v případě, kdy při reprodukci reálných podmínek používání a zatížení na základě doby odezvy jsou nalezena slabá místa.
- *Testování vytrvalosti* ověřuje stabilitu systému a jeho výkon při dlouhodobém běhu. Jeho cílem je odhalit případné úniky paměti, snížení výkonu nebo zvýšení latence či omezení škálovatelnosti.
- *Testování přerušení* vyhodnocuje, jak aplikace reaguje na neočekávaná přerušení. V závislosti na povaze přerušení by se aplikace měla buď zastavit a vrátit do původního stavu, nebo reagovat určitým způsobem.

Tato forma testování je jednou z nejdůležitějších a lze jí získat identifikaci slabých míst, optimalizaci zdrojů a samotného kódu, zlepšení uživatelského zážitku a zajištění škálovatelnosti. Na druhou stranu může být potřeba specifických nástrojů pro měření, testy mohou být finančně a časově náročné a vyžaduje komplexnější znalosti pro analýzu výsledků a jejich správnou opravu. [45][46]

2 HERNÍ PRŮMYSL

Od konce 70. let 20. století, s nástupem titulů jako Pong a Pac-Man, začal herní průmysl pronikat do širšího povědomí veřejnosti. Po více než padesáti letech se stal jedním z nejdynamičtěji rostoucích odvětví zábavního průmyslu s odhadovanou tržní hodnotou kolem 200 miliard dolarů. Současně se videohry etablovaly jako klíčový kulturní fenomén, který formuje způsoby trávení volného času i mezilidské interakce. [10][47]

Kromě samotných vývojářů a testerů zaměstnává herní sektor široké spektrum specialistů – od hudebních skladatelů přes herní designéry až po scénáristy a herce. Díky neustálému technickému pokroku navíc posouvá hranice hardwaru a inovuje platformy pro hraní či streamování. Mezi nejoblíbenější herní platformy patří PC, PlayStation, Xbox a Nintendo, zatímco pro streamování dominují služby YouTube a Twitch. [10][47]

2.1 Vyvíjené platformy

Jak již bylo zmíněno, herní průmysl je jedním z nejrychleji rostoucích odvětví, která se vyvíjejí ruku v ruce s technologickým světem. Aktuálně se hry vyvíjí a distribuují na několik hlavních platform – PC, konzole, Cloud, VR/AR a mobilní zařízení. Tyto platformy se liší nejen hardwarovými a softwarovými parametry, ale i způsobem distribuce, monetizací a cílovou uživatelskou základnou. [10][47]

2.1.1 PC

Osobní počítače představují nejrozšířenější a nejoblíbenější platformu pro hraní videoher, neboť kombinují širokou dostupnost s vysoce flexibilním nastavením. V roce 2023 se GDC (Game Developers Conference) zeptala vývojářů, pro kterou platformu aktuálně vyvíjí projekt a 65 % odpovědělo pro PC. Platforma je výhodná zejména pro Indie vývojáře, pro které je vývoj her na PC snazší než pro konzole, a v některých případech i než pro iOS a Android. [48].

Hry pro PC mohou být díky většímu přizpůsobení HW více graficky propracované. Jejich vývoj ale je komplikovanější právě díky různým variacím HW jako jsou CPU Intel vs. AMD, GPU NVIDIA vs. AMD nebo různé OS jako Windows, MacOS a Linux. V návaznosti na žánr hry a vyvíjený OS a HW testování obsahuje prakticky vše, co se běžně testuje v rámci funkcionálních a nefunkcionálních testů. Testování ale může být komplikovanější právě kvůli větší paletě HW a SW konfigurací. [7][47]

2.1.2 Konzole

Konzole jsou standardizovaný systém, který nabízí hráčům hraní her v rámci jejich uzavřeného ekosystému. Mohou vznikat exkluzivní hry, které například na PC nebo jiných konzolích nejsou, anebo mohou nabízet exkluzivní funkce jako haptickou odezvu v ovladači DualSense od PlayStationu nebo režim myši na konzoli Nintendo Switch 2. Mezi nejpoužívanější konzole patří PlayStation 5 od Sony, Xbox Series X od Microsoftu a Nintendo Switch (a jeho nástupce Switch 2) od Nintenda. Tito dodavatelé nabízí vývojářům své SDK (Software development kit) a nástroje pro vývoj, kompilaci a testování, za které musí vývojář většinou zaplatit. Výhodou těchto zařízení je stabilní HW, pro který je často snazší hru optimalizovat.

Podle výsledků z GDC je druhou nejvíce používanou platformou pro vývoj her PlayStation 5 s 33 %. Následuje Xbox Series X/S s 30 % a na sedmém místě je Nintendo Switch s 18 % [48]. Oproti testování na PC je zde více regulací a kromě všech testů jako na PC (funkčnost, lokalizace, výkon) je nutné projít i certifikačním procesem – například TRC (Technical Requirements Checklist) pro Sony nebo XR (Xbox Requirements) pro Microsoft. Nedodržení může znamenat zamítnutí publikace na dané platformě. Kromě tohoto certifikačního procesu musí hráči mít tzv. devkity konzolí, které jsou přizpůsobeny právě pro vývoj hry. [47][49][50]

2.1.3 Cloud

Cloudové služby zajišťují hraní her skrze internet, přičemž je hra reálně spuštěna na serverech a hráč si pouze přenáší obraz a zvuk na své zařízení. Mezi největší služby patří Xbox Cloud Gaming, NVIDIA GeForce NOW a PlayStation Plus. Velkou výhodou je placení paušální částky za celou knihovnu her, pro kterou není potřeba mít výkonný HW, ale pouze obrazovku, ovladač/klávesnici s myši a přístup k rychlému internetu. Hry se tak můžou hrát na televizi, mobilu, tabletu i PC. Nevýhodou je ale závislost na vysokorychlostním internetu a stabilita sítě. Podle výsledků dotazníku GDC z roku 2023 plánuje 7 % vývojářů dát svůj příští titul na Xbox Cloud Gaming a 4 % na PlayStation Plus Extra nebo Premium [48]. Testování pro tuto oblast představuje hlavně zaměření na síťové aspekty a kontrolu latence, kvality obrazu při různých rychlostech připojení, schopnosti obnovy při výpadku a uživatelských vstupů. Klíčovým faktorem je zde tedy testování infrastruktury a streamovací vrstvy. [47][51]

2.1.4 VR/AR

Hráč využívá virtuální realitu (VR) tehdy, když chce být plně ponořen do světa kompletně vytvořeným počítačem. Na druhou stranu rozšířená realita (AR) přináší do reálného světa virtuální prvky jako třeba postavy nebo UI elementy. V obou případech musí mít hráč na hlavě helmu nebo brýle, které pomocí miniaturních displejů a kamer přenášejí virtuální obraz do oka, kde jej hráč zpracuje. [47][52]

Pro GDC se přes 36 % vývojářů vyjádřilo, že jejich projekt do virtuální reality by nejradši dali do brýlí Meta Quest. 18 % by využilo VR headsetu PS VR2 od PlayStationu. První dvě místa jsou stejná i pro jejich aktuální projekty [48]. Vývoj i testování je pro tyto platformy více odlišný a přináší výzvy. Hry musí být optimalizovány pro minimálně 90 FPS s co nejnižší latencí, aby u hráčů nedocházelo k tzv. kinetóze. Tedy nevolnost způsobenou odlišným vnímáním pohybu těla a vizuálních vjemů. [53]

Kromě těchto testů se musí dávat i větší pozornost na výdrž baterie, kontrola více typů ovládání (např. ruce, ovladače nebo hlas) a zohlednění vlivů reálného prostředí jako světelnost v místnosti nebo její velikost. Je důležité též kontrolovat UI elementy, jejich rozložení v prostoru, kontrolu v případě překročení bezpečné meze pro hraní a schopnost obnovy při výpadku některého ze senzorů. Mezi nejoblíbenější VR hry se řadí Beat Saber a pro AR to je Pokémon Go. [47][52]

2.1.5 Mobilní platformy

Díky své snadné dostupnosti a výběru pro každého jsou mobilní hry velmi populární a tvoří přibližně 45 % celkového podílu v herním průmyslu. Mezi nevýhody patří omezená výdrž baterie, úložiště, velikost obrazovky a výkonu. Stejně jako u PC i zde je široká škála HW periférií a OS k pokrytí (nejčastěji různé verze Androidu a iOS). Vývojáři v dotazníku GDC až ve 27 % případů vyvíjeli aktuální projekt pro Android a 26 % pro iOS. Po PC a konzolách aktuální generace jsou tedy mobilní platformy na čtvrtém a pátém místě v nejpoužívanějších platformách pro hry [48]. Při testování je třeba zohlednit různé operační systémy a jejich verze, chování hry při přerušení (hovor, notifikace), funkčnost in-app nákupů a reklam, integraci se službami Google Play či Apple Game Center a ověřit vše na reálných zařízeních. [2][47]

2.2 Profese

V herním průmyslu je často nutné obsadit specializované role, které jinde obvykle nejsou potřeba. V závislosti na složení týmu může jeden člověk zastávat více funkcí současně, avšak vždy zůstává hlavním cílem dodat úspěšný produkt včas a v rámci rozpočtu, aniž by byl ztrátový či obestřen kontroverzí. Tyto cíle lze naplnit pouze díky efektivní týmové spolupráci. [4][54]

První důležitou skupinou jsou **programátoři**. Hrají klíčovou roli při převodu kreativních nápadů do interaktivních a funkčních herních prvků. Obvykle se specializují na konkrétní oblasti vývoje, které dohromady tvoří technologické jádro celé hry. Mezi tyto oblasti patří programování herní logiky a mechanik (gameplay programátoři), tvorba a optimalizace vizuálního zobrazení (render programátoři), simulace fyzikálních jevů (fyzikální programátoři) a vývoj umělé inteligence, která oživuje herní svět (AI programátoři). [4][54]

V případě online a multiplayerových her jsou nepostradatelní síťoví a backend programátoři, kteří zajišťují komunikaci mezi hráči a správu serverového prostředí. Specifickou roli pak hrají programátoři nástrojů a systémů. Ti vytvářejí interní nástroje, které zefektivňují práci celého týmu, a přizpůsobují herní engine požadavkům konkrétního projektu. [4][54]

Vývojáři nejčastěji vytváří a debugují kód a opravují chyby. Využívají herní enginy jako Unity, Unreal Engine a CryEngine a často pracují s jazyky C++, C# nebo shaderovými jazyky HLSL a GLSL. Úzce spolupracují se zbylými členy týmu (testéři, herní designéři, grafici atd.) pro zlepšení celkové kvality softwaru. [4][54]

Druhou skupinou jsou **designéři**. Ti formují kreativní vizi hry a celkový její zážitek. Navrhují herní mechaniky, rovnováhu mezi nimi, příběh, úrovně a herní ekonomiku. Vytváří herní dokumentaci a stejně jako programátoři i designéři se dělí do několika skupin. Level designér navrhuje úrovně a prostředí, které hráči projdou a testují jejich dovednosti. UX/UI designéři se zaměřují na uživatelské prvky ve hře a jejich chování. Tato oblast se zabývá přehledností menu, HUD (Head-up display) a ovládání s cílem zajistit nerušený herní zážitek a plynulou interakci. Technický designér propojuje návrhy s implementací za pomoci vizuálního skriptování a zajišťuje správné fungování herních prvků. Systémoví designéři se soustředí na herní mechanismy jako ekonomika, bojové mechaniky či postup hráče. Narativní designéři navrhují celkovou atmosféru ve hře a vytváří příběh, dialogy a vývoj

postav. V případě hudby, efektů a prostorového zvuku se pracuje se zvukovými designéry. [4][54]

Další klíčovou skupinou, na kterou se tato diplomová práce do značné míry zaměřuje, jsou **testeři (Quality Assurance)**. Jejich úkolem je zajistit, aby byl výsledný produkt zábavný, funkční a stabilní. Zkoumají hru z různých aspektů, identifikují chyby, vytváří testovací strategie, připravují si testovací data, kontrolují specifikaci a komunikují různé chyby nebo vylepšení s týmem. Testeři se mohou specializovat v různých oblastech, i když často mají spíše širší záběr. Ověřují funkční testy, kompatibilitu, výkonnost, testy použitelnosti a testy dodržování předpisů (compliance testing). [4][54]

Animátoři a umělci jsou důležití po stránce audio vizuálu. Umělecký tým přispívá k estetice, atmosféře a obchodnímu úspěchu finálního softwaru. Dělí se na 3D grafiky, kteří vytváří modely různých objektů a postav v nástrojích jako Blender nebo Maya a zajišťují správné fungování a vzhled v herním enginu. Koncept a 2D artisté navrhují celkový vizuální styl hry skrze ilustrace, které používá celý tým pro lepší představu např. mechaniky nebo postavy. Riggeři vytvářejí kostry a ovládací prvky pro 3D modely, aby se mohly realisticky pohybovat během animace. [4][54]

Animátoři rozpohybují postavy a objekty pomocí vytvořených animací, přičemž pracují v herních enginech (např. Unity, Unreal Engine) nebo v nástrojích jako Blender. Marketingoví grafici pak připravují upoutávky, bannery a obaly s cílem zaujmout co nejširší okruh hráčů a efektivně hru propagovat. [4][54]

Poslední klíčovou skupinou je **management**. Ten se stará především o to, aby byl projekt dokončen včas, v souladu se stanoveným termínem a nastavenou tvůrčí vizí. Projektoví manažeři koordinují, plánují a sledují průběh prací napříč týmy za využití nástrojů, jako jsou Trello, Jira nebo Asana. Producenti kombinují obchodní a kreativní hlediska. Zodpovídají za rozpočet, alokaci zdrojů a plnění kvalitativních i finančních cílů projektu. Dále kreativní ředitelé dohlíží na celkovou uměleckou vizi hry a na zajištění jejího správného naplnění. Kontrolují příběh, mechaniky, vizuální styl i hudbu a snaží se ve všem najít takový průnik, aby výsledek byl co nejlepší. [4][54]

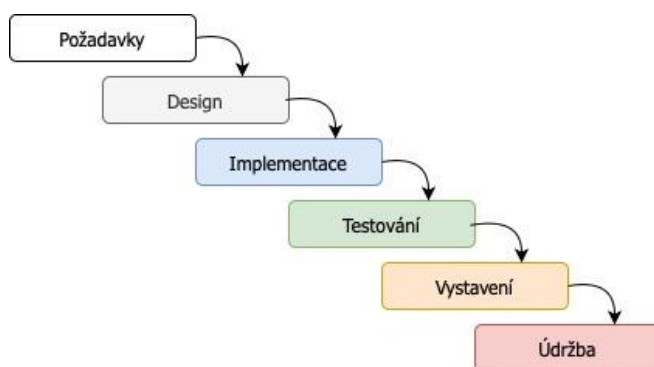
Technický ředitel řeší složité technické výzvy a dohlíží na správné fungování hry na koncových platformách. Komunitní manažer zprostředkovává komunikaci mezi týmem a herní komunitou. Sleduje reakce hráčů, odpovídá na jejich dotazy, informuje o novinkách a snaží se vytvořit co nejvíce pozitivní vztah mezi hráči a vyvíjenou hrou. [4][54]

2.3 Životní cyklus

Životní cyklus vývoje softwaru je definován různými pravidly a modely, které určují strukturu, pořadí a způsob realizace jednotlivých fází. Obecně je tento postup označován jako Software Development Life Cycle (SDLC). V herním odvětví je proces nazýván Game Development Life Cycle (GDLC) a je rozdělen do tří fází – preprodukce, produkce a postprodukce. Každé fázi jsou přidělena specifika, přičemž je kladen maximální důraz na systematický přístup a pečlivé plánování. V této kapitole jsou porovnávány tradiční modely scrum, vodopád a spirála i specializované modely určené pro herní průmysl. Pozornost je věnována rozdílům mezi iterativním a lineárním přístupem a na základě specifik týmu je vybrán nejvhodnější model. [54][56][57]

2.3.1 Vodopádový model (Waterfall model)

Jedna z nejstarších metodik byla představena v 70. letech 20. století. Primární využití bylo zaznamenáno u projektů, v nichž byl kladen důraz na plánování, dokumentaci a předvídatelnost vývoje. Model je definován sekvenčním průchodem jednotlivými fázemi vývoje, které byly stanoveny v pevně daném pořadí a na sebe navazují. To vyobrazuje Obrázek 5. [56][58]



Obrázek 5 Vodopádový model

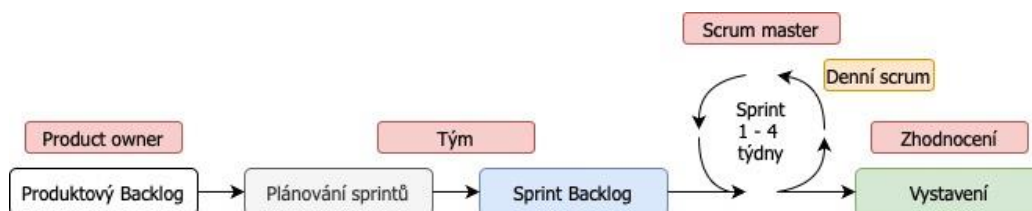
(zdroj: vlastní)

Každá vývojová fáze musí být zcela dokončena před přechodem do fáze následující. Změny ve fázích již uzavřených jsou obtížně proveditelné a často vyžadují návrat procesu o několik kroků zpět, což generuje vysoké časové i finanční náklady. Model je proto vhodný především pro projekty s předem definovanými a neměnnými požadavky, například interní firemní aplikace se striktně danými procesy. [56][58]

V kontextu herního odvětví vodopádový model naráží na mnoho omezení a jeho použití se obecně nedoporučuje. Dochází k situacím, kdy jsou různé chyby či nevyvážené herní mechaniky objeveny až v pozdních fázích, takže jejich oprava je ekonomicky neefektivní. Například při sekvenčním postupu skrze analýzu, návrh a implementaci kompletních úrovní se zásadní problémy v herních mechanikách (pohyb hráče, interakce objektů, propady snímků apod.) objeví až během testování poté, co byly všechny předchozí fáze uzavřeny. V té chvíli jsou nutné úpravy vysoce komplikované – termíny musí být dodrženy a vývojáři s grafikou jsou již obvykle přiděleni na jiné projekty, protože jejich účast na daném projektu byla oficiálně ukončena. Naopak při uplatnění iterativních přístupů, jako je scrum, je možné tyto problémy odhalit dříve a opravy provést efektivněji. Z uvedeného vyplývá, že vodopádový model nachází uplatnění jen v projektech s předem jasně definovanými a stabilními požadavky, pro herní vývoj je však většinou nevhodný.[56][58]

2.3.2 Scrum

Scrum patří mezi nejrozšířenější agilní metodiky současnosti. Jeho vznik se datuje do 90. let 20. století jako reakce na nedostatečně flexibilní přístupy (například vodopádový model), které nevyhovovaly měnícím se požadavkům. Obrázek 6 znázorňuje vizuální schéma tohoto modelu. [56][59]



Obrázek 6 Scrum metoda

(zdroj: vlastní)

Scrum klade důraz na iterativní vývoj, při němž je software vytvářen postupně v krátkých časových úsecích (sprinty) trvajících 1–4 týdny. Každému sprintu je přiřazen jasně definovaný cíl a vzniká verze produktu připravená k nasazení. Během sprintu se aktivně podílejí všechny zainteresované strany, přičemž je klíčová transparentnost postupu práce a efektivní komunikace v rámci týmu. V rámci Scrum frameworku jsou vymezeny tři základní role: Product owner, který odpovídá za řízení požadavků, maximalizaci hodnoty produktu, prioritizaci úkolů a správu backlogu. Scrum master, jenž zajišťuje organizaci procesu, odstraňuje překážky a dohlíží na dodržování scrum pravidel. A vývojový tým, který v průběhu sprintu realizuje konkrétní úkoly a dodává funkční přírůstky produktu. [56][59]

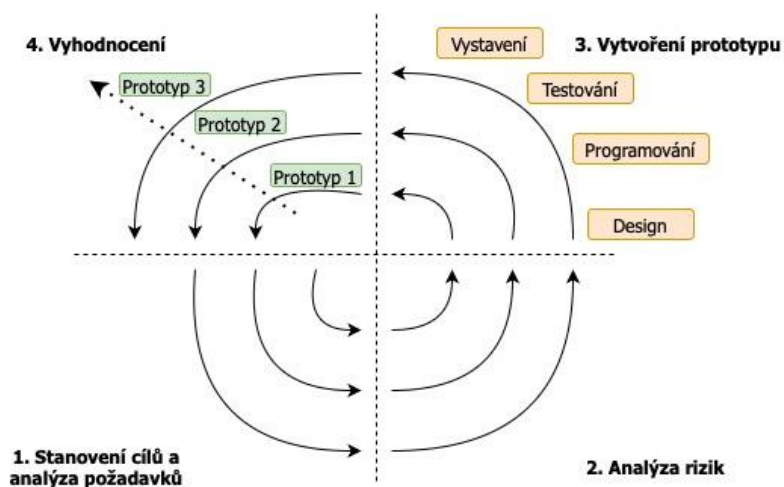
Následně jsou mezi klíčové události scrumu definovány: [56][59]

- *Product backlog a Sprint backlog* – soubory požadavků (nebo jejich vybraných podmnožin) určených k realizaci v rámci jednoho sprintu. Spravuje je Product Owner.
- *Inkrement* – verze produktu připravená k otestování a dalšímu nasazení v rámci ukončeného sprintu.
- *Daily scrum* – každodenní setkání zaměřené na synchronizaci týmu a včasnou identifikaci překážek.

Pro herní vývoj se scrum osvědčil jako vysoce efektivní přístup. Při adekvátním přizpůsobení umožňuje týmu rychle odstraňovat překážky a pružně reagovat na změny. Klíčové je ale upravit proces tak, aby respektoval různorodé časové nároky jednotlivých disciplín (např. tvorba assetů vs. programování), což jinak může ztížit plánování sprintů. [56][59]

2.3.3 Spirálový model (Spiral model)

Metodika, ve které jsou kombinovány prvky vodopádového přístupu s iterativním vývojem, je charakterizována důrazem na řízení rizik. Představení této metodiky proběhlo v roce 1988 jako reakce na omezenou použitelnost lineárních přístupů zejména ve velkých a komplexních projektech. Vývoj je organizován do postupně rozšiřujících se cyklů (spirál), přičemž každý cyklus představuje samostatnou iteraci a do další spirály jsou zařazeny stále podrobnější fáze plánování, vývoje a testování. [56][60][61]



Obrázek 7 Spirálový model

(zdroj: vlastní)

Obrázek 7 znázorňuje model. Z obrázku je patrné, že každý cyklus spirály zahrnuje čtyři hlavní kvadranty: [56][60][61]

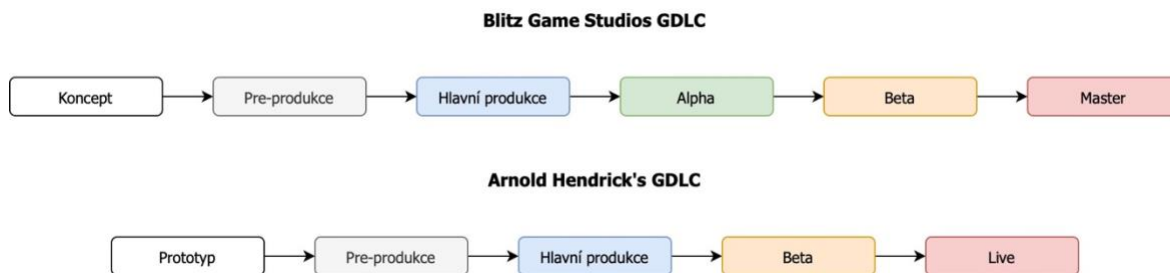
- Analýza požadavků a návrh možných řešení, při níž jsou stanoveny cíle a alternativy.
- Identifikace a analýza rizik, klíčová fáze pro stanovení strategie jejich řešení.
- Vytvoření návrhu, jeho implementace a testování prototypu.
- Vyhodnocení výsledků a návrh dalšího postupu.

Tento přístup oproti vodopádovému modelu umožňuje opakované zhodnocení produktu na základě výstupů každé iterace, obsahuje návrhy na vylepšení a adaptaci na nové požadavky. Díky tomu je proces flexibilnější, usnadňuje řízení rizik a průběžné testování i validaci produktu. [56][60][61]

V herním průmyslu lze tento model uplatnit zejména v rozsáhlých, inovativních a technologicky náročných projektech. V praxi se jeho využití prokazuje při opakovaném návrhu a testování herního prototypu. V první iteraci je vytvořen úvodní prototyp. Ve druhé iteraci jsou integrovány nové prvky, jako vylepšené ovládání či detailnější textury. V každém dalším cyklu jsou stávající komponenty optimalizovány a rozšiřovány, přičemž je vždy provedena analýza rizik (např. identifikace nedostatečné optimalizace nebo nevyvážených herních mechanik). Další uplatnění model nachází při současném vývoji hry a herního enginu nebo v projektech s otevřeným světem a dynamickými systémy. Jako nevýhoda bývá uváděna zvýšená komplexita řízení projektu a v menších týmech vnímání modelu jako nadměrně robustní. [56][60][61]

2.3.4 Lineární a iterativní modely navržené speciálně pro herní průmysl

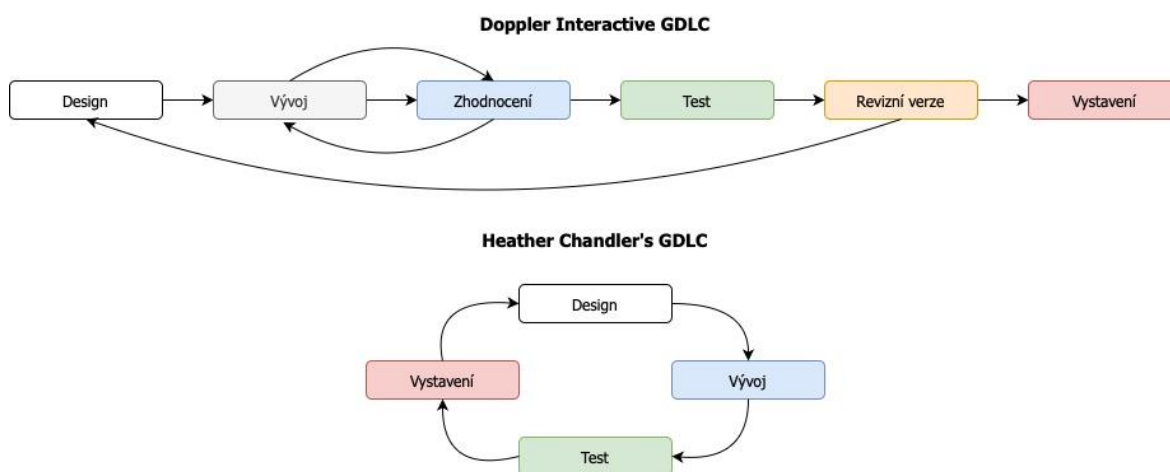
Obrázek 8 a Obrázek 9 je porovnání čtyř modelů vývoje v herním odvětví, přičemž barevné rozlišení je využito k označení fází – bílá je přiřazena návrhu a prototypu včetně tvorby herní dokumentace a definice mechanik, ovládání a vizuální podoby. Šedá a modrá jsou vyhrazeny vývoji a jeho průběžnému zhodnocení. Zelená a oranžová jsou určeny internímu a externímu testování a červená pak značí finální nasazení hry do produkce.[57]



Obrázek 8 Porovnání lineárně herních specializovaných metod

(zdroj: vlastní)

Modely Blitz Game Studios a Arnolda Hendrickse jsou oba řazeny mezi lineární přístupy (Obrázek 8). U modelu Blitz Game Studios je zavedena podrobnější fáze vývoje a testování, čímž je na obě části kladen větší důraz. Naproti tomu u Arnoldovy metodiky je interní testování vynecháno a produkt je předáván přímo k externímu testování (externí společnosti nebo hráčům v rámci Early Access). Lineární přístup je obecně nedoporučován kvůli častým změnovým požadavkům a je využíván pouze u titulů s pevně stanovenými požadavky a menším týmem. [57]



Obrázek 9 Porovnání iterativních herních specializovaných metod

(zdroj: vlastní)

Naopak Obrázek 9 zobrazuje iterativní modely přizpůsobeny hernímu vývoji tak, že umožňují pružné úpravy produktu podle aktuálních potřeb. U modelu Doppler Interactive je kladen důraz na cyklické vyhodnocování každé verze, přičemž je díky zpětné smyčce z revize verze na fázi designu zajištěna vysoká flexibilita při úpravách a sníženy náklady na změny. V modelu Heather Chandlera je dovoleno opakované přecházení mezi návrhem, vývojem a testováním bez navazujících schvalovacích kroků, čímž je dosaženo výrazného zrychlení vývojového procesu.[57]

2.4 Testování techniky

Je využíván jako nástroj pro návrh efektivní testovací strategie a zajištění širokého spektra testů během vývoje softwaru. Každá technika je uplatněna v příslušné fázi vývoje nebo pro specifický typ problému a nachází uplatnění jak v tradičním softwarovém vývoji, tak ve vývoji her. Mezi tyto techniky patří kombinatorické testování, testovací vývojové diagramy, cleanroom testování, testovací stromy a ad hoc testování včetně playtestingu. Při vhodné kombinaci jednotlivých metod je dosaženo zvýšení kvality softwaru při současném snížení nákladů a zkrácení času.

2.4.1 Kombinatorické testování

Kombinatorické testování je zaměřeno na efektivní pokrytí různých kombinací vstupních parametrů tak, aby byl minimalizován počet testovacích případů při zachování schopnosti odhalit chyby. Čas i náklady jsou šetřeny vhodně navrženými kombinatorickými testy, které se uplatňují zejména u ovládacích menu, filtrů a dalších interakčních prvků aplikací. S narůstajícím počtem parametrů je pro automatické generování optimálních sad kombinací využíváno nástrojů, jako jsou NIST ACTS nebo Hexawise. Nejčastěji jsou používány dva přístupy: [7][62]

- Úplné kombinatorické testování, při kterém jsou testovány všechny možné kombinace hodnot všech parametrů – tento přístup je principiálně nejspolehlivější, avšak s rostoucím počtem parametrů rychle narůstá i počet testovacích případů.
- *Párové (pairwise) testování*, které zajišťuje, aby se každá dvojice hodnot ze dvou různých parametrů objevila alespoň jednou, čímž je počet testů výrazně snížen bez podstatné ztráty schopnosti odhalovat chyby.

Parametry jsou rozlišovány do několika typů. *Výchozí hodnoty* jsou považovány za nejčastěji používané konfigurace. *Výčty (enumerace)* představují diskrétní množiny možností, z nichž by měla být každá zastoupena v testech. *Rozsahy* označují numerické hodnoty obvykle reprezentované posuvníky. Kladen je důraz především na hraniční hodnoty (min, max, nula). *Hraniční hodnoty* definují časové nebo prostorové limity (např. konec mapy, délka mise).

Při sestavování párové tabulky jsou nejprve vybrány parametry s největším počtem hodnot a vytvořen jejich sloupec. Hodnoty dalšího parametru jsou doplňovány tak, aby bylo zajištěno maximální pokrytí dvojic s předchozím parametrem. Každý následující parametr

je do tabulky zařazen postupně. Pro každý řádek je vybrána hodnota, která maximalizuje počet nově pokrytých dvojic. Po dokončení vyplňování je ověřeno, zda jsou všechny možné páry zahrnuty, a v případě nedostatků jsou příslušné řádky přidány nebo upraveny [7][62]

Kritické mechanismy se testují úplně, zatímco ostatní funkce jsou pokryty párově. Při 10 % kritických funkcích (100 kombinací) a 90 % nekritických (20 testů) je dosaženo snížení počtu testů z 100 N (kde N představuje celkový počet parametrů) na 28 N, což představuje úsporu 72 % u nekritických částí. [7][62]

Za zvláště přínosné je považováno použití této metody u složitých aplikací nebo aplikací obsahujících desítky vstupních dat. V konečném důsledku je široký rozsah testování pokryt menším počtem testovacích případů, čímž jsou sníženy náklady i čas.

2.4.2 Testovací vývojové diagramy

Grafické modely TVD jsou navrženy pro zachycení herního chování z pohledu hráče. Pro návrh modulárních, opakovatelných testovacích scénářů jsou využívány a kontrola mezi týmy je jimi zjednodušena. Mezi komponenty TVD jsou zařazeny: [7][62][63]

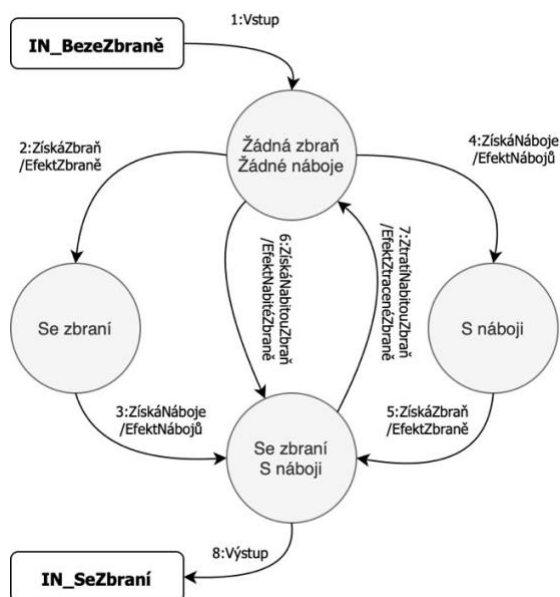
- *Terminátory (IN/OUT)* jsou využívány k vyznačení začátku a konce testu. Každý diagram je jimi vybaven.
- *Stavy (State)* jsou považovány za stabilní herní situace a vykreslují se jako bubliny. Každému stavu je přiřazen alespoň jeden vstupní a jeden výstupní tok, přičemž duplicitní stavy jsou konsolidovány.
- *Toky (Flow)* jsou definovány jako propojení dvou stavů, obsahují přesně jednu událost a jednu akci a jsou označeny unikátním ID.
- *Událost (Event)* je iniciována hráčem, sítí, zařízením nebo hrou (např. spuštění dialogu). Každá událost je v toku zastoupena právě jednou, avšak může být opakována v různých tocích.
- *Akce (Action)* je reakcí hry na událost (např. vizuální efekt, zvuk nebo haptická odezva) a je vždy spárována s konkrétní událostí.
- *Primitiva* jsou definována v datovém slovníku a představují základní názvy a popisy událostí, akcí a stavů.

Nejprve jsou shromážděny a identifikovány všechny požadavky a herní prvky, které spadají do rozsahu testování – mise, schopnosti, objekty či libovolné interakční prvky. Následně je sestaven úplný seznam položek, jež mají být v diagramech zachyceny. [7][62][63]

Následně je rozsah rozdělen na samostatné diagramy a odhadnut jejich počet. Velké množiny prvků jsou rozkládány do menších logických bloků, které jsou buď podle schopností (každé schopnosti je přiřazen vlastní diagram), nebo podle dosažitelných cílů (např. mise, výzva, hádanka). Tím je zajištěno lepší zvládnutí komplexity a umožněna paralelní práce více testerů. [7][62][63]

V závěrečné fázi jsou TVD sestavena a doplněna o detailní definice a testovací cesty. Průběh od IN do OUT je nejprve nakreslen, přičemž jsou vloženy jednotlivé stavy, toky událostí a odpovídající akce v očekávaném pořadí. [7][62][63]

Po dokončení diagramů je vytvořen centrální datový slovník, v němž jsou jednoznačně definovány všechny primitiva (události, akce a stavy) používaná napříč TVD. Z diagramů jsou následně generovány konkrétní testovací cesty v podobě posloupností $IN \rightarrow Event \rightarrow Action \rightarrow State \rightarrow \dots \rightarrow OUT$, které jsou přepsány do textových skriptů nebo přímo integrovány do automatizovaných testů prostřednictvím kódových instrukcí. Výstup diagramů je prezentován v podobě Obrázek 10. [7][62][63]



Obrázek 10 Zjednodušená reprezentace TVD

(zdroj: vlastní)

2.4.3 Cleanroom testování

Cleanroom testování vychází z přístupu Cleanroom Software Engineering, v němž je kladen primární cíl na minimalizaci chyb již ve fázi vývoje a na měření střední doby do selhání (MTTF). V herním průmyslu je tento model využíván spíše k realistické simulaci reálného hraní než k izolovanému odhalování chyb. Hra je aktivně hraná s cílem zachytit situace, které se mohou u hráčů skutečně objevit, a cleanroom je tak začleňován do širší testovací strategie společně s funkcionálním a regresním testováním. [64]

Klíčovým nástrojem je *model pravděpodobnosti použití*. Pro každou herní funkci jsou váhy přiřazeny podle očekávané četnosti jejího využití hráči. Tyto váhy jsou odvozeny z výzkumu hráčského chování, analytických dat a předpokládaného herního postupu. Například délka a obtížnost závodů je zvyšována s rostoucí úrovní hráče. Díky tomu je simulace přiblížena reálnému nasazení a výkon systému je objektivně sledován. [7][64]

Měření kvality softwaru je prováděno především pomocí ukazatele *MTTF (Mean Time To Failure)*, se vzorcem: [65]

$$MTTF = \frac{\text{Celková doba provozu}}{\text{Počet poruch}} \quad (2)$$

Průměrná doba bezchybných běhů je tímto parametrem monitorována za účelem vyhodnocení stability produktu a predikce budoucích selhání. V rámci cleanroom testování jsou pro simulaci hráče analyzovány tři hlavní dimenze použití: [7][64][65]

- *Režim hry* – různé módy (kampaně, multiplayer, survival) jsou podle svých odlišných vzorců hraní realisticky modelovány.
- *Typ hráče* – profily (nováček, příležitostný, hardcore) jsou odlišeny na základě preferencí (průzkum, achievementy, sociální interakce, boj).
- *Reálné nasazení* – skutečné chování hráčů je sledováno analytickými nástroji, přičemž nasbíraná data z live provozu slouží k doladění simulačních modelů.

2.4.4 Testovací stromy

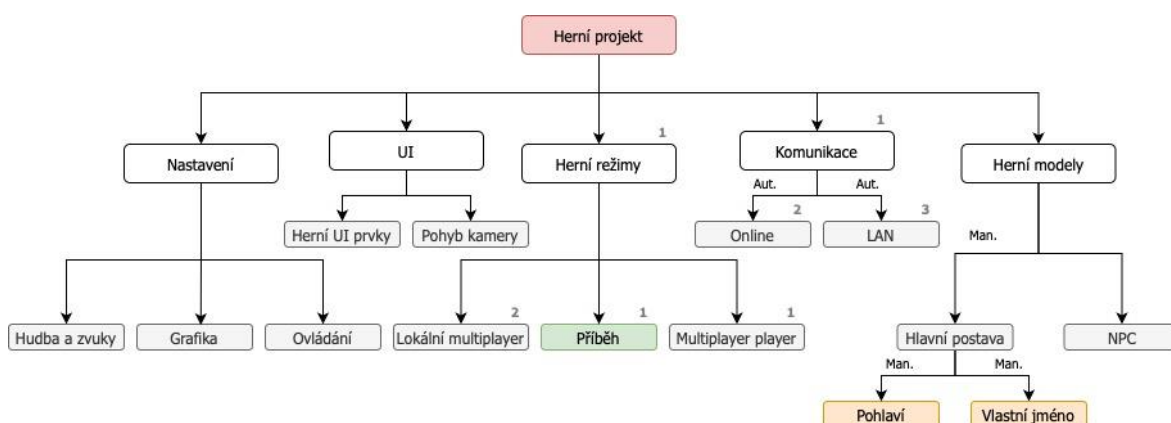
Testovací stromy jsou navrženy jako strukturovaná hierarchie, v níž jsou přehledně zobrazeny vztahy mezi herními prvky, funkcemi a vlastnostmi. Větve stromu mohou být barevně označeny (např. zeleně pro nové funkcionality, oranžově pro netestované části) a očíslovány podle priority. Hlavní oblasti využití testovacích stromů jsou: [7][66]

Organizace a správa existujících testů – Jakmile jsou testovací případy definovány, je hierarchie jejich vztahů podle herních mechanik (režimy, mapy, UI aj.) zobrazena v testovacím stromu. Pomocí této struktury je zajištěno cílené vyhledání relevantních testů pro konkrétní změny v kódu, garantuje se požadované pokrytí a minimalizuje riziko opomenutí scénářů.[7][66]

Návrh nových testů – Při vizualizaci oblastí určených k otestování jsou v testovacím stromu odhalena „prázdná místa“ bez existujícího pokrytí. Na základě zkušeností testerů jsou do stromu doplňovány chybějící větve reprezentující nové kombinace či varianty a okamžitě se provádí aktualizace struktury. [7][66]

Zobrazení závislostí stávajících a nových prvků – Hierarchie a předpoklady pro aktivaci dalších prvků (např. stromy dovedností, odemykání map či úrovní) jsou v testovacím stromu znázorněny tak, aby bylo možné otestovat pozitivní průchody (odemknutí) i negativní scénáře (pokus o využití nepřístupné funkce). Pro každou cestu jsou vytvářeny testovací matice, v nichž jsou ověřovány nejen správné průchody, ale i adekvátní zacházení s chybovými stavy. [7][66]

Obrázek 11 je znázorněním obecné podoby testovacího stromu. Zelené políčko označuje novou funkcionalitu určenou k přidání, oranžové políčko pak funkcionalitu dosud netestovanou. Popisky „Aut.“ a „Man.“ u větví slouží k rozlišení automatizovaných a manuálních testů. Nad jednotlivými políčky jsou uvedena čísla 1 až 3, která vyjadřují priority (1 nejvyšší, 3 nejnižší).



Obrázek 11 Příklad testovacího stromu

(zdroj: vlastní)

2.4.5 Ad Hoc a Play testování

Ad Hoc testování je považováno za nestrukturovaný přístup, kterým je umožněno improvizovat a odhalovat chyby unikající formálním metodám testování. Obvykle je prováděno bez detailního plánování a dokumentace a je nasazováno až po dokončení formálního testování. Jeho cílem je odhalit obtížně reprodukovatelné chyby a otestovat krajní scénáře (např. pokus o výstup na objekty, kam přístup není povolen). Klíčovým opatřením je předcházení skupinovému myšlení, přičemž je stanovování individuálních cílů považováno za nezbytné k udržení efektivity. [67][68]

V rámci **play testování** jsou hodnoceny subjektivní kvality hry – zábavnost, vyváženost, obtížnost a celkový herní zážitek. Hra je přehrávána testery či skupinou uživatelů, kteří poskytují detailní zpětnou vazbu zaměřenou na intuitivnost ovládání, plynulost a náročnost. Testování je realizováno v libovolné fázi životního cyklu, nejčastěji však v předběžné fázi, kdy je hra poskytnuta hráčům, jež vyplňují dotazníky, hlásí zjištěné chyby a hodnotí produkt čerstvým pohledem. [67][69]

II. PRAKTICKÁ ČÁST

3 TESTOVANÁ HRA PRESERVE

V této práci budou výsledky prezentovány na hře Preserve [70], která byla vyvinuta slovenským studiem Bitmap Galaxy [71] se sídlem v Bratislavě ve spolupráci s produkční a vydavatelskou společností Grindstone [72] z Košic. Hra byla 8. 8. 2024 vydána do předběžného přístupu na platformě Steam a zpřístupněna na Windows, Mac a SteamOS/Linux. Konzolové varianty jsou rovněž vyvíjeny, avšak nebudou předmětem této práce. K březnu 2025 bylo shromážděno 315 velmi pozitivních recenzí.

Preserve je logickou relaxační hrou pro jednoho hráče, v níž je hráči umožněno budovat vlastní harmonický ekosystém s cílem udržet v rovnováze různorodé biomy. Karty jsou vykládány tak, aby prostředí upravovaly či rozšiřovaly, a následně jsou do něj vkládány živočichové a rostliny.



Obrázek 12 Ukázka ze hry

(zdroj: vlastní)

Jsou definovány tři herní režimy. V režimu harmonie jsou odměny přidělovány na základě nasbíraných bodů harmonie prostřednictvím přidávání habitatů, živočichů, přírodních divů a rozšiřování mapy. V kreativním režimu je umožněno neomezené vytváření vlastních map a ekosystémů bez jakýchkoli omezení. V režimu hádanek je hráči předložena pevně daná mapa s kartami a cílem je dosáhnout požadovaného počtu bodů harmonie pro úspěšné dokončení úrovně.

3.1 Herní logika

Ve hře jsou definovány čtyři odlišné biomy – kontinentální, savana, mořský a prehistorický¹. Každý z biomů je charakterizován jedinečnou sadou rostlin, živočichů, environmentálních výzev a mechanik. Herní mapy i karty jsou podle biomu rozlišeny.



Obrázek 13 UI hry

(zdroj: vlastní)

3.1.1 Herní UI

Uživatelské prostředí je rozděleno na čtyři části (viz Obrázek 13). Červená oblast je vymezena jako **ovládací panel**, jehož prostřednictvím se vyvolává hlavní nabídka, spouští fotografický režim, přepíná se zobrazení UI habitatu a otevírá se bodovací listina.

Oranžová část představuje **panel akcí**, v němž jsou zpřístupněny funkce pro přidání nových částí mapy, zobrazení požadovaného a aktuálního skóre, krok zpět a ukončení úrovně po splnění zadání.

Zelená oblast vymezuje **herní mapu**, na níž jsou šestiúhelníkové dlaždice reprezentující environmentální prvky, jež fungují jako stavební kameny pro přidávání

¹ Testování bylo provedeno v době, kdy prehistorický biom nebyl k dispozici, a proto na něj není v práci kladen důraz.

habitatů či živočichů. Dlaždice jsou nabízeny ve třech terénních výškách (nížina, vysočina a hory pro kontinentální biom) a pro každý biom jsou jedinečné.

Poslední část tvoří **karty hráče**, ve kterých jsou ve formě informací o typu habitatů, povolené terénní výšce a počtu kusů definovány možnosti úprav herní plochy.

3.1.2 Bodovací systém

Body harmonie jsou udělovány za umístění terénních, zvířecích a speciálních karet. Karty jsou přiřazeny konkrétnímu biomu a jejich bodové ohodnocení je určováno na základě umístění i kombinace s ostatními kartami. Níže je uvedeno bodové hodnocení.

Tabulka 1 je rozdělena do čtyř částí. V první části jsou uvedeny základní bodové akce společné pro všechny biomy – body za zvlhčení půdy (kontinentální a savana biom) a za vyvýšení terénu (mořský biom). Dále jsou v ní definovány body za vytvoření habitatu a za vložení zvířat. Například bylo vypočítáno, že za vytvoření habitatu z 12 dlaždic se třemi různými zvířaty bylo uděleno celkem 114 bodů: 24 bodů za dlaždice, 30 bodů za vložení zvířat a 60 bodů jako bonus za tři rozdílná zvířata.

Tabulka 1 Bodovací tabulka

Akce	Celkový počet bodů
Malá karta deště/vyvýšení terénu	1
Velká karta deště/vyvýšení terénu	7
3 dlaždicový habitat s 1 zvířetem	16
6 dlaždicový habitat s 2 zvířetem	32
12 dlaždicový habitat s 3 stejnými zvířaty	84
12 dlaždicový habitat s 3 různými zvířaty	114
Kontinentální biom	
Řeka	+1 za dlaždici řeky +1 za každou sousední dlaždici s biotopem
Jezero	+1 za každou sousední dlaždici +1 za každou sousední dlaždici s habitatem
Vodní mlýn	+2 za každou sladkovodní plochu na mapě
Přírodní divy	+10 za každou dlaždici daného biotopu
Mořský biom	
4 stejná zvířata (přírodní div)	80 -> 30 + 50 bonus

4 různá zvířata (přírodní div)	110 -> 30 + 80 bonus
Maják	+2 za každý ostrov na mapě
Termální průduch	+1 bod za dlaždici průduchu V závislosti na počtu stejného habitatu kolem průduchu se přičtou další body. Pokud je tedy kolem průduchu 6 stejných dlaždic samého habitatu, dostane +6 bodů za každou dlaždici.
Ostrov	+1 za dlaždici ostrova +1 za ostrov do velikosti 2 dlaždic +2 za ostrov do velikosti 5 dlaždic +5 za ostrov od velikosti 6 dlaždic
Savana biom	
Řeka	+1 za dlaždici řeky +1 za řeku do velikosti 11 dlaždic +2 za řeku do velikosti 23 dlaždic +5 za řeku od velikosti 24 dlaždic
Jezero	+1 za každou sousední dlaždici +1 za jezero do velikosti 2 dlaždic +2 za jezero do velikosti 5 dlaždic +5 za jezero od velikosti 6 dlaždic
Dům na chůdách	+2 za každou dlaždici s čerstvou vodou
Přírodní divy	+30 za každou sousední dlaždici daného habitatu

(zdroj: odvozeno z pravidel)

Pro kontinentální biom jsou v tabulce definovány body za přidání řeky a jezera, která zavlažují okolní dlaždice. Body jsou udělovány jak za samotnou kartu, tak za okolní dlaždice. Například za 18 dlaždic dlouhou řeku bylo přiděleno až 75 bodů (34 bodů za kartu řeky, 1 bod za její počátek a 40 bodů za okolní dlaždice). Dále jsou stanoveny body za karty vodního mlýna a přírodních divů – jezero s vodním mlýnem bylo ohodnoceno 32 body (2 body za jezero a 30 bodů za okolní dlaždice) a přírodní div v kombinaci s 12 dlaždicovým habitatem a třemi různými zvířaty dosáhl 234 bodů (144 bodů za dlaždice, 30 bodů za zvířata a 60 bodů bonusu).

Pro mořský biom je definováno, že majákem je hodnota každého ostrova navýšena o 2 body. Je-li na mapě šest ostrovů, může být dosaženo 118 bodů (6 bodů za ostrovy a $16 \cdot 7$ bodů za okolní habitaty). Při přidání majáku je za každý ostrov uděleno 9 bodů místo 7, což umožňuje celkové skóre až 150 bodů. Umístěním přírodního divu je dále povoleno zahrnout do habitatu čtyři zvířata místo tří.

V biomu Savana je opět umožněno umístění řeky nebo jezera. Navíc je možné využít Dům na chůdách, kterým jsou všem sladkovodním dlaždicím přidány 2 body. Bodové ohodnocení bylo stanoveno takto: za jezero obklopené šesti dlaždicemi habitatu bylo přiděleno 18 bodů ($6 \cdot 3$ body) a po přidání Domu na chůdách byla hodnota každé dlaždice zvýšena na 5 bodů, čímž bylo dosaženo celkového skóre 30 bodů ($6 \cdot 5$). U přírodního divu je za každou sousední dlaždici habitatu přidáváno 30 bodů. Při dvanácti okolních dlaždicích bylo výsledné skóre vypočteno na 384 bodů.

4 TESTOVACÍ SCÉNÁŘE

V herním studiu Bitmap Galaxy jsou zaměstnání pouze vývojáři, u nichž dosud nebyly sepsány žádné testovací scénáře (TS) ani dokumentace. Mezi službami, jež jsou poskytovány vydavatelem, je zahrnuto i testování. Na základě diskuse s Lead QA bylo určeno, že hlavním úkolem je řešení situace hráče, který hru právě zakoupil na Steamu. V rámci této práce byly vytvořeny testovací scénáře (TS) v Microsoft Excelu, které jsou dostupné v příloze diplomové práce. Scénáře mohou být využity jak k ověření funkčnosti, tak i jako návod k plnění hádanek či k rychlejšímu osvojení herních mechanik.

Bylo stanoveno několik pravidel, která byla při tvorbě TS dodržována:

- Jelikož část týmu nemluví česky nebo slovensky, všude byla **použita angličtina**.
- Všechny TS byly **doplňeny o obrázky**, pokud to bylo přínosné.
- U každého testu by měla být zajištěna možnost **nastavení stavu** (úspěšně/neúspěšně).
- TS by měly být rozdělené podle jednotlivých částí hry.
- Měla by být použita co nejvíce **univerzální formulace**, aby byl význam ihned pochopitelný a usnadněna aktualizace.
- Byl použit nástroj na zápis TS, který je co nejvíce rozšířený a snadný na pochopení.

Obrázek 14 reprezentuje jeden z vytvořených TS v Excelu. Karta byla rozdělena do čtyř částí, které byly na obrázku zvýrazněny barevně. Oranžová část je hlavička a čtenáři poskytl krátký popis a uvedení do dané sekce ve hře.

Name of the test scenario: Continental Storm - Puzzle			
Description of the mechanism of the first puzzle in the Continental Storm section			
Test scenario objectives			
Title: Continental Storm - Puzzle mode			
ID	Description	Pass	Fail
T0001	Check cards	When bottom of the screen in the middle is a card, include 20 cards for 1 day. 12 forest cards, 10 deer cards, 10 wolf cards and 10 rabbit cards.	Pass
T0002	Check the score	The bottom left of the screen shows the current score and the desired score.	High
T0003	Test the puzzle board	Place the 12 forest cards on the green tiles (there can be added to existing or already green tiles). Verify that a player receives 20 points for correctly placing a forest card on a green tile.	Pass
		Place the 10 deer cards in the habitat created with the forest. Verify that the player receives 10 points for each deer.	Pass
		Use the 10 wolf cards and verify that the player can return one animal or forest card back and move it. The points are returned along with the card.	Pass
		Verify that after all these actions, the player achieves a total score of 117 points as shown in the last puzzle.	Pass
T0004	Successful completion of the puzzle	When the required number of points has been earned, a final message will appear on the screen.	Pass
T0005	Testing error and boundary conditions	After clicking, a window will appear where you can go to the next puzzle or go back to the main menu. To place the forest card on the green tile, verify that the system will not allow this action and the player will not lose any points. To use the wolf cards, verify that the system blocks this action and the player loses no points. Attempt to place an animal card on a tile that is not part of a habitat (place 10 forest cards on the map). Verify that the system does not accept the card and the player loses no points. Attempt to use a recycling card on an empty tile with no forest or animal cards. Verify that the card has no effect and the player loses no points. Try to add more animal cards to a habitat than the	Pass

Obrázek 14 Popis TS

(zdroj: vlastní)

Druhá část (červená) je tvořena tabulkou s rozepsanými testovacími případy (TC), v níž jsou uváděny názvy TC, jejich popis, kroky, priorita (nízká, střední, vysoká), stav průchodu (zaznamenává se, zda daný krok byl úspěšně proveden) a poznámka.

Ve třetí části (zelené) byly umístěny obrázky, které byly zobrazeny tam, kde může jejich přítomnost usnadnit pochopení testovacího scénáře. Obrázek 14 je doplněn o obrázky k hádance, kdy první je určen k reprezentaci počátečního stavu a druhý k zobrazení dokončené hádanky. Díky tomu je testerovi umožněno okamžitě přistoupit k požadovanému kroku bez nutnosti řešení hádanky. V poslední části (fialové) byly uvedeny poznámky k mechanikám.

Jako ukázka do diplomové práce je uveden následující příklad testovacího scénáře pro tutoriál. V případě, že budou některé testovací scénáře doplněny o navazující metody testování (např. kombinatorická tabulka), budou tyto metody popsány v příslušné podkapitole.

4.1 Tutoriál

Testovací scénář popsal začátek průchodu tutoriálu a vysvětlení základních mechanik ve hře. V Excel souboru *TS – Functional tests* byla vytvořena stránka s 23 testovacími případy, které jsou uvedeny níže. Obrázek 15 je využit k znázornění hlavičky testovacího scénáře, v níž je uveden název, popis a navigační cesta z hlavního menu k tutoriálu.

Name of the test scenario:	<i>Tutorial</i>
Test scenario objective:	<i>Basic description of the tutorial section behavior</i>
Path:	Play -> Tutorial

Obrázek 15 Hlavička testovacího scénáře pro Tutoriál

(zdroj: vlastní)

V poznámkách k testovacímu scénáři (Obrázek 16) jsou obsaženy popisy dostupných karet. Jsou zde uvedeny karty malého a velkého zavlažení dlaždic, karty lesa a karty divočáka.

Notes on Mechanics:
Small rain Card: Water 1 clay tiles, turning them green and adding 1 point for each tile watered.
Large rain Card: Water 7 clay tiles, turning them green and adding 1 point for each tile watered.
Forest card: can only be placed on green/watered tiles. When creating a habitat (3 side by side), one card is worth 2 points.
Boar Card: May be placed in a habitat of forests or wetlands (3 cards side by side). Each animal adds 10 points.

Obrázek 16 Poznámky k testovacímu scénáři Tutoriálu

(zdroj: vlastní)

ID	Description	Steps	Priority	Status
TC001	The welcome window opens	click on Okay	Low	PASSED
TC002	A description for moving around the map appears in the top bar	try the W,A,S,D keys or left click After moving the top bar disappears	Medium	FAILED
TC003	A description for rotating or tilting the map appears in the top bar	try Q, E or right/left mouse button to rotate try R,F or right mouse button up/down to tilt After rotation/tilt the top bar disappears	High	PAUSED
TC004	A description for zooming the map appears in the top bar	try T,G keys or mouse wheel up/down to zoom After zooming the top bar disappears	Immediate	PASSED
TC005	A window appears describing the number of cards	click on Okay	Low	FAILED
TC006	Shows a top bar describing the function of the rain cards, a playing area on which the 3 tiles to apply the rain card are highlighted, and a separate highlighted rain card at the bottom	I left-click on the rain tab. The tab will be highlighted Hover the mouse over the board. Rain can only be applied to the highlighted tiles. When clicked, an animation starts where it rains and the tiles turn green. Points will be added to the overall score. This will be done 3 times	Low	PASSED
TC007	A window describing the harmony points is displayed.	I click on Okay	Low	FAILED
TC008	A window describing the harmony milestones appears.	click on Okay	Low	FAILED
TC009	A top bar describing the function of the forest cards, the playing area and, at the bottom, a separate highlighted forest card will appear	I left-click on the forest card. The card will be highlighted I hover the mouse over the board. Forest can only be applied to grass tiles When clicked, an animation will start where a tree will grow Points are added to the total score. This will be done 3 times	Low	FAILED

Obrázek 17 Tabulka s testovacími případy pro Tutoriál

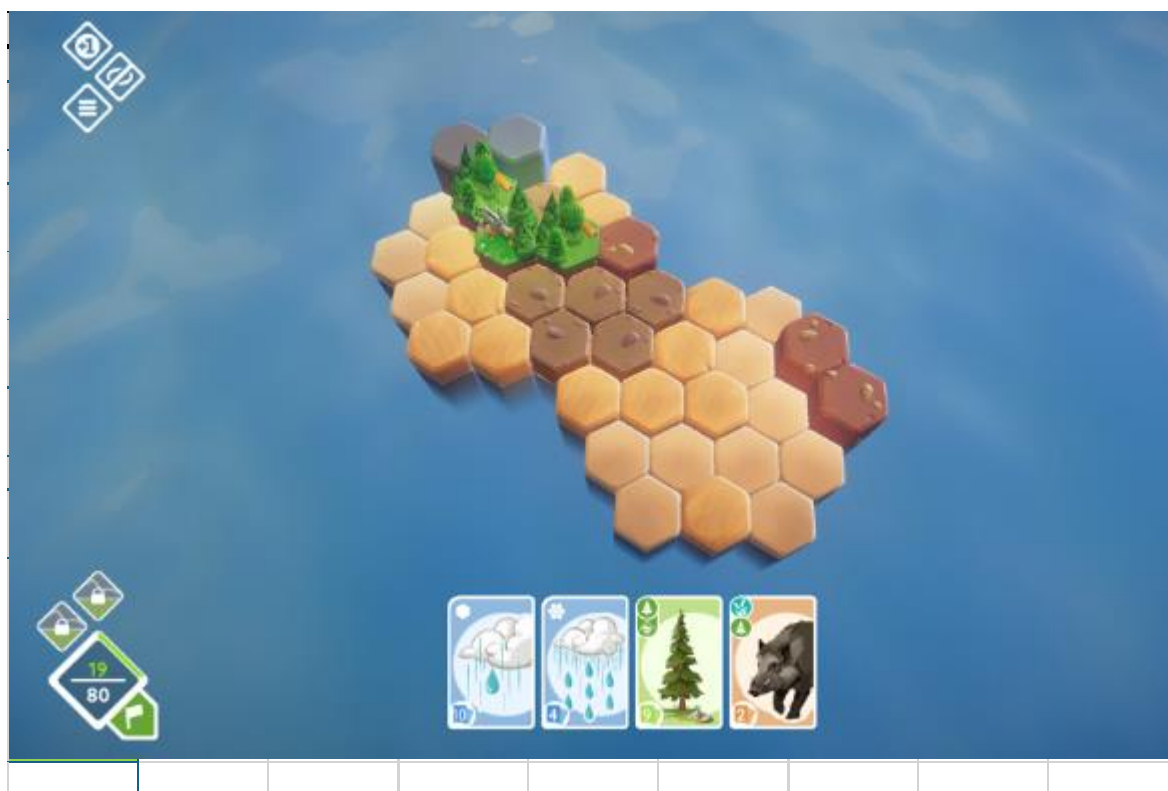
(zdroj: vlastní)

Uvedena tabulka (Obrázek 17), v níž je zobrazeno 9 z celkových 23 testovacích případů pro tutoriál. Testovací případy TUT_TC001 a TUT_TC002 jsou zaměřeny na vysvětlení pohybu po herním plátně. V TUT_TC003 je popsáno otáčení pomocí kláves Q, E, R a F (doleva, doprava, nahoru, dolů).

Přibližování bylo objasněno v TUT_TC004 s využitím tlačítek T, G nebo kolečka myši. Základy karet a chování zavlažovacích karet jsou specifikovány v TUT_TC005

a TUT_TC006. Body harmonie a milníky harmonie byly rozebrány v TUT_TC007 a TUT_TC008. Chování lesních karet bylo podrobně popsáno v TUT_TC009 (šest kroků). A v TUT_TC010 a TUT_TC011 je prezentováno zobrazení okna pro vytvoření habitatu spolu s ikonami na kartách.

Možnosti přidání divokého prasete do biomu lesa jsou v TUT_TC012 popsány v šesti krocích. Okno s body za zvířata a následné možnosti rozšíření mapy v herním plánu jsou představeny v TUT_TC013 a TUT_TC014. Testovací případy TUT_TC015, až TUT_TC017 jsou určeny ke kontrole tabulky skóre, zobrazení okna ukončení tutoriálu a otevření nastavení pro volnou hru či ukončení tutoriálu. TUT_NTC001 až TUT_NTC006 jsou určeny k negativnímu testování.



Obrázek 18 Konec tutoriálu

(zdroj: vlastní)

Obrázek 18 zobrazuje vizualizaci ukončení tutoriálu, po jehož dokončení je přechod do režimu volné hry umožněn. Uživatelům byla poskytnuta volba, zda bude hra opuštěna prostřednictvím vlaječky, nebo v ní bude pokračováno. Výsledek této sekvence může být testerem ověřen a následně může být pokračováno v testování dalších oblastí.

4.2 Ovládání

Na třech záložkách jsou testovací scénáře zaměřeny na ověření správného ovládání, mapování jednotlivých tlačítek a nastavení ovládacího schématu pro klávesnici s myší a Xbox ovladač. Podpora PlayStation DualShock 4 a DualSense byla vyloučena.

Kromě klasické podoby testovacích scénářů, obsahujících standardní tabulky s testovacími případy, hlavičku TS a obrázky sekcí, je v případě konfigurace ovládání využita i kombinatorická tabulka (Obrázek 19). Tento heterogenní test byl navržen tak, aby maximálně podrobně pokryl možnosti nastavení ovládání: z teoretických $3^3 \cdot 2^4$ (432) kombinací bylo vybráno devět testů, jež zajišťují pokrytí většiny scénářů.

Combinatorial table								
	Movement speed	Rotation/tilt speed	Controller cursor speed	Invert X camera	Invert Y camera	Invert zoom	Invert tilt	Status
UC1	1	1	1	Yes	Yes	Yes	Yes	
UC2	5	5	5	No	No	Yes	no	
UC3	10	10	10	Yes	No	No	yes	
UC4	1	5	10	No	Yes	No	no	
UC5	5	10	1	No	Yes	No	yes	
UC6	10	1	5	No	Yes	No	no	
UC7	1	10	5	Yes	No	Yes	yes	
UC8	5	1	10	Yes	No	Yes	no	
UC9	10	5	1	Yes	No	Yes	yes	

Obrázek 19 Kombinatorická tabulka nastavení ovládání

(zdroj: vlastní)

5 TESTOVÁNÍ APLIKACE

Testování her bylo provedeno na třech zařízeních představujících různé třídy výkonu: ASUS ROG Ally Z1 Extreme jako přenosné vysoce výkonné PC s Windows, MacBook Air M1 jako střední třída s macOS, Acer Aspire E1-531 jako velmi zastaralé PC a HP ProBook 450 G8 jako méně výkonné PC. HW specifiky sestav byly následující:

HP ProBook 450 G5 byl vybaven Windows 11 Pro i Ubuntu 24.04.1 LTS a osazen 4jádrovým procesorem Intel Core i7-8550U (1,8 GHz základ, až 4,0 GHz boost), integrovanou grafikou Intel UHD Graphics 620, 16 GB RAM a 15" Full HD displejem s obnovovací frekvencí 60 Hz (SDR).

Acer Aspire E1-531 byl vybaven operačním systémem Ubuntu 24.04.1 LTS, dvoujádrovým procesorem Intel Pentium B960, integrovanou grafickou kartou HD Graphics 2000, 6 GB RAM a 15,6" displejem s HD rozlišením.

MacBook Air byl osazen 8jádrovým SoC čipem M1 s 3,2 GHz základním taktem, integrovanou 8jádrovou GPU a 16 GB sdílené paměti, a je vybaven 13" Retina displejem s podporou sRGB a obnovovací frekvencí 60 Hz.

ASUS ROG Ally Z1 Extreme byla navržena jako handheld konzole s předinstalovaným Windows 11, osazena 8jádrovým procesorem AMD Ryzen Z1 Extreme s základním taktem 3,3 GHz (boost až 5,1 GHz), integrovanou grafickou kartou Radeon 780M s architekturou RDNA 3.0, 16 GB paměti s frekvencí 1500–2500 MHz a dotykovým 7" Full HD IPS displejem s podporou sRGB a obnovovací frekvencí 120 Hz.

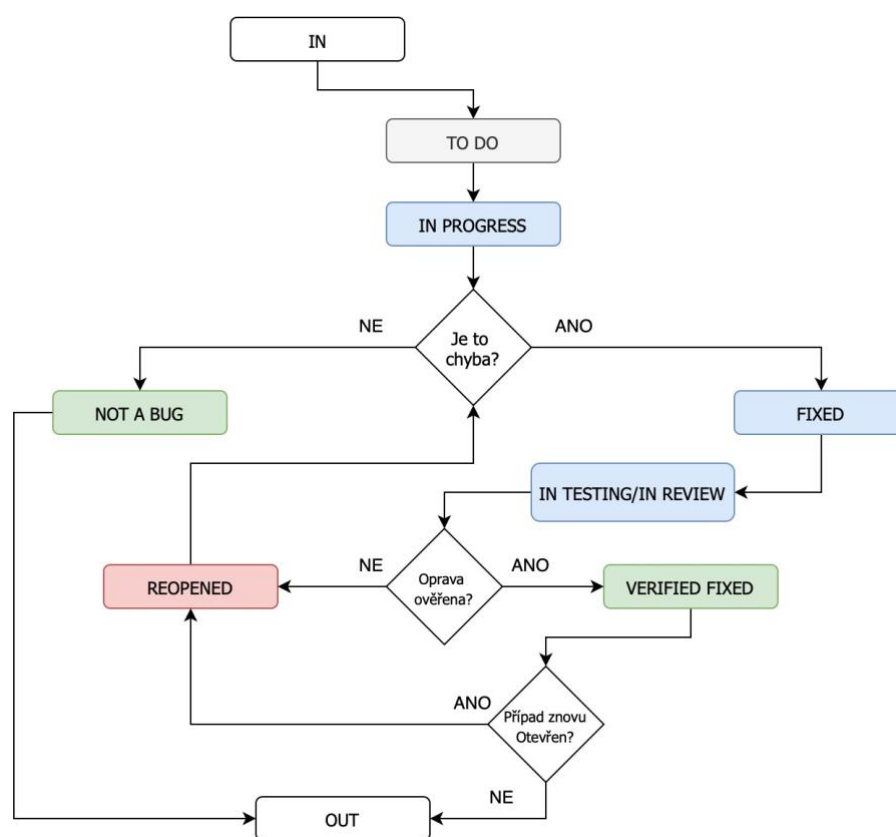
5.1 Zapisování nalezených chyb

Při objevení chyby byl v Jiře založen ticket, do něhož byly vloženy co nejpodrobnější a zároveň nejsrozumitelnější informace o chybě. Následně byl ticket předán vývojářům, kteří po opravě ticket zpět odeslali testerovi k ověření. V případě úspěšné opravy byl ticket označen jako „opravený“ a uzavřen. Pokud se v chybném chování pokračovalo, byl ticket s doplněným popisem opět předán vývojářům. Obrázek 21 ilustruje příklad takového ticketu.

Při zakládání ticketu je vytvořen výstižný název, podle kterého je určena základní příčina. Pro lepší organizaci je jako první uváděn název sekce, v níž se chyba projevila. Jako příklad z obrázku může být použit následující název:

The menu screen background gets stuck after quickly pressing B button

Název je formulován tak, aby v ucelené a jednoduché formě obsahoval informaci o místě výskytu i o povaze chyby, a umožnil tak rychlou orientaci dalším účastníkům procesu. Typ ticketu (nejčastěji bug) a priorita jsou určovány na základě závažnosti chyby, pravděpodobnosti její reprodukce a termínových omezení. Jako klíčové atributy jsou uvedeny verze, ve které se chyba projevila, a reprodukční kroky, které jsou očíslovány a navádějí vývojáře k opětovnému vyvolání problému. Sprint je obvykle doplňován QA leadem nebo analytikem, přičemž počáteční status ticketu je nastaven na TO DO (Obrázek 20 znázorňuje celý životní cyklus ticketu).



Obrázek 20 Životní cyklus ticketu v Jiře

(zdroj: vlastní)

Pro popis bylo vymezeno další důležité políčko, do kterého byl zaznamenán podrobný popis problému a chybového stavu. V případě potřeby byl přiložen odkaz na specifikaci. K dokumentaci chyby byla připojena příloha (obrázek nebo video). Při pořízení videa bylo využito rozšíření pro OBS Studio, jež umožnilo na obrazovce zobrazit stisknuté klávesy či tlačítka myši/ovladače. Na závěr byly všechny informace ověřeny z hlediska úplnosti a správnosti a ticket byl předán vedoucímu vývojářského týmu k dalšímu zpracování.

Po provedení analýzy chyby vývojářem byl stav ticketu změněn na IN PROGRESS a následně v případech, kdy se neprokázalo, že se jednalo o bug, na NOT A BUG. V ostatních případech na FIXED. Ticket byl poté přiřazen buď testerovi, který chybu původně nahlásil, nebo jinému dostupnému členovi týmu. V případě úspěšné opravy byl stav ticketu aktualizován na VERIFIED FIXED, v opačném případě byl ticket opět otevřen (REOPEN) nebo doplněn o další kroky k řešení.

The screenshot shows a Jira ticket interface. At the top, the ticket title is "The menu screen background gets stuck after quickly pressing B button" with a status of "FIXED (View Workflow)". The ticket is assigned to Jorge Rosique and reported by Zdenek Pohludka. The details section includes: Type: Bug, Priority: Medium, Labels: None, Severity: Medium, Build Version: 0.6.5.g, and Repro Steps: 1. go to options, 2. visit control -> control settings, 3. change something, 4. go back to game by pressing the B button. The description states: "When you change something on settings and go back to game quickly pushing the B button, you get stuck on menu screen background. It's not happening every time, but quite often." The attachments section shows a video file named "2024-12-03 20-10-10.mkv" with a timestamp of "03/Dec/24 10:37 PM" and a size of "9.00 MB". The Slack section indicates that access to the workspace "Games Farm" is required to see discussions.

Obrázek 21 Ticket Jiry s nalezenou chybou

(zdroj: vlastní)

5.2 Funkcionální testování (Functional testing)

V této části byly vytvořeny testovací scénáře, podle kterých bylo následně testováno obecné chování hry. Byly ověřovány aspekty jako chování jednotlivých biomů, funkčnost tutoriálu, průchodnost herních módů, správnost bodovacího systému, fungování základních prvků (karty, dlaždice), funkčnost menu a jeho UI/UX, splnitelnost jednotlivých hádanek a ovládání pomocí ovladače či klávesnice s myší. Nalezené chyby byly různého charakteru – od nefunkčních tlačítek kamery až po nemožnost dokončení hádanky bez využití chyby.

5.2.1 Biomy – chování základních prvků

Ve Příloze souboru *TS – Functional tests* jsou tři záložky pojmenovány podle jednotlivých biomů. Každá je zaměřena na ověřování funkčnosti herních prvků a mechanik (např. úrovní elevace či zavlažování, biotopů, živočichů a speciálních prvků, jako jsou přírodní divy). V rámci třiceti tří testovacích případů je zahrnuta validace správnosti animací, zvukových efektů, bodovacího systému a interakcí mezi herními kartami a dlaždicemi. Všechny scénáře ve verzi 0.6.70 byly úspěšně dokončeny a vyhodnoceny jako splněné.

5.2.2 Tutorial

V souboru *TS – Functional tests* je k dispozici záložka Tutorial, ve které je ve 23 testovacích případech ověřeno fungování tutoriálu ve hře. V rámci testů jsou zahrnuta zobrazení úvodních oken, popisy ovládání a herních funkcí, interakce s herními prvky (karty deště, lesa a zvířat) včetně animací a přidávání bodů, stejně jako informace o skóre, milnících a rozšíření mapy. Dále je testováno ukončení tutoriálu a přechod do volné hry. Všechny testovací případy pro verzi 0.6.70 byly úspěšně dokončeny a žádné chyby nebyly zjištěny. Tato sekce byla jako jedna z prvních vytvořených již optimalizována.

5.2.3 Herní módy

Testovací scénář je umístěn v záložce *Game modes* souboru *TS – Functional tests* a zahrnuje čtyři klíčové herní módy: puzzle, tutoriál, klasickou hru a kreativní mód. V režimu puzzle je ověřována splnitelnost úrovní, jasnost cílů, pevný počet karet a možnost restartu po neúspěchu, přičemž je kladen důraz i na rozmanitost výzev (např. zvětšení mapy či omezený počet karet). Tutoriál je testován z hlediska přístupnosti z hlavního menu, správného představení základních mechanik a poskytování nápovědy při nečekaných akcích hráče. V klasickém módu je kontrolováno správné bodování, ukončení hry při vyčerpání karet a zachování strategického charakteru. V kreativním módu je ověřován neomezený přístup ke kartám a dlaždicím, funkčnost herních mechanik, možnost restartu mapy a korektní ukládání a načítání hry. Všechny scénáře ve verzi 0.6.70 byly úspěšně dokončeny a vyhodnoceny jako splněné.

5.2.4 Herní bodování

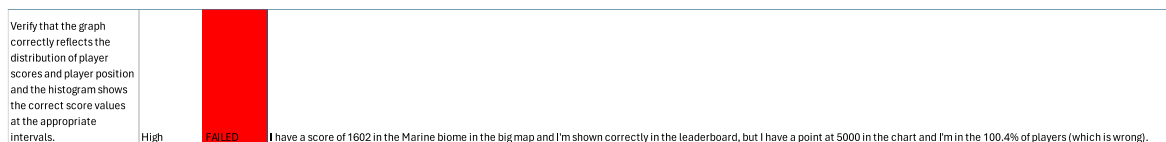
V testovacím scénáři hry byla ověřována funkčnost bodovacího systému herních mechanik, zahrnující použití karet pro úpravu terénu (zavlažení, zvýšení terénu, řeky, termální průduchy aj.), tvorbu biotopů, přidávání staveb (vodní mlýny, majáky) a implementaci

přírodních divů. Byly testovány různé podmínky, včetně maximálních a minimálních limitů mechanik, vzájemné interakce herních prvků a správného přidělování bodů za jednotlivé akce. Všechny testy verze 0.6.70 byly úspěšně dokončeny a bylo potvrzeno, že herní mechaniky fungují dle specifikací. Scénář je umístěn v záložce *Game points logic* souboru *TS – Functional tests*.

5.2.5 Herní prostředí

V rámci testování tohoto scénáře, který je zaměřen na ověření chování karet, recyklace herních prvků, funkčnosti herního prostředí, ovládacích a akčních panelů, ukládání a načítání hry, systému notifikací (včetně bodů, výzev a harmonie) a funkčnosti žebříčků v různých režimech, bylo stanoveno deset testovacích případů. Ve všech případech byly testy úspěšně dokončeny s výjimkou funkčnosti žebříčků. Testovací scénář je umístěn na záložce *Game environment* v souboru *TS – Functional tests*.

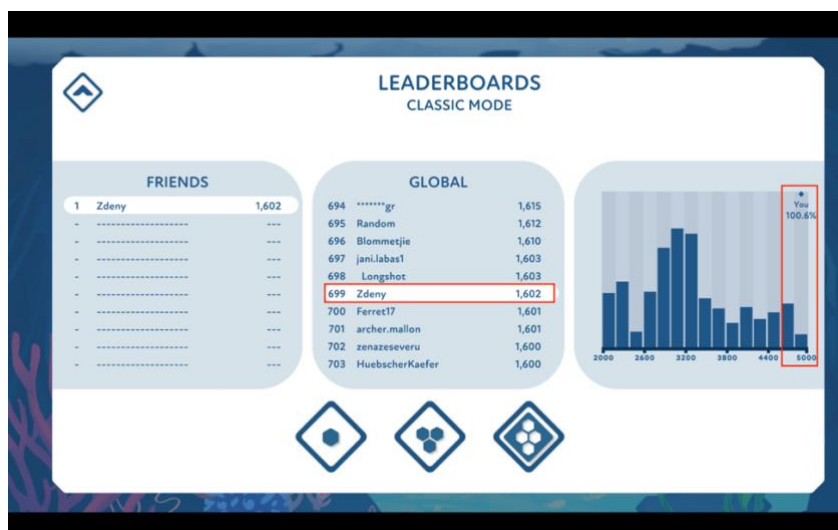
První detekovaná chyba byla zaznamenána ve 4. a 7. kroku (jde o tentýž problém). V tomto případě byla v nastavení pro největší mapu mořského biomu na grafu zobrazená nesprávná pozice vybraného hráče vůči žebříčku. Chyba je podrobně popsána v TS na řádce 64 (viz Obrázek 22).



Obrázek 22 GE_TC010 – Chybné zobrazení v grafu vůči žebříčku

(zdroj: vlastní)

Obrázek 23 vyobrazuje tuto chybu. Pozice hráče *Zdeny*, která byla identifikována jako 699. s nejvyšším skórem 1602 bodů, byla v grafu na pravé straně umístěna nad hodnotou 5000 a označena procentem 100,6, což neodpovídá reálné pozici. Správné zobrazení by mělo bod umístit na levé straně grafu před 2000 body, případně jej z grafu zcela odstranit.



Obrázek 23 Chyba v TC010

(zdroj: vlastní)

Druhá chyba menšího charakteru byla zaznamenána v situaci, kdy není navázáno připojení k internetu a po kliknutí na žebříčky se data s grafem a pozice hráče načítají neustále (viz Obrázek 24). V takovém případě by bylo vhodné, aby po uplynutí určitého časového intervalu nebo na základě konkrétního HTTP statusu bylo zobrazeno hlášení typu *Unable to show current leaderboard stats*. Výskyt této chyby je pochopitelný vzhledem k nedávnému přidání dané funkcionality.

Turn off your internet connection and try to load the leaderboard. An error message will appear indicating that the data cannot be loaded.	Medium	FAILED	Its just loading and nothing more. It should be better to add some text like "Unable to show current leaderboard stats"
--	--------	--------	---

Obrázek 24 GE_TC010 – Doplnění informace o nedostupnosti žebříčku

(zdroj: vlastní)

5.2.6 Menu, nastavení a UI

V záložce *Menus, settings and UI* souboru *TS – Functional tests* jsou uvedeny testovací případy zaměřené na základní prvky hry. Jednotlivé sekce jsou barevně odlišeny. V prvních třech testovacích případech je ověřováno UI interních prvků a hlavního menu. Ve dvou následujících je validován fotografický režim. V dalších dvou je testováno ukončení a zahájení hraní. V rámci dalších 22 testovacích případů bylo komplexně ověřeno nastavení hry (s výjimkou sekcí lokalizace a ovládání, pro něž byly využity samostatné soubory s testovacími scénáři) a v posledních třech případech bylo otestováno ukládání a načítání hry v kreativním režimu.

Pro ověření audia ve hře byla rovněž sestavena kombinatorická tabulka, která umožnila snížit počet testovacích případů z 3^4 (81 scénářů) na 9. Touto tabulkou byly pokryty nejpravděpodobnější chyby v interakci mezi různými posuvníky, přičemž kromě jediné závady bylo testování absolvováno bez dalších problémů. V testovacím kroku MSUI_TC011 (Obrázek 25) byla odhalena chyba – po ukončení přehrávání skladby v pozadí uplynuly přibližně tři minuty, než byla spuštěna skladba následující. Tato chyba byla zaznamenána pouze v mořském biomu.

Check that the music is playing in a loop and does not stop playing if it is on.	Medium	FAILED	In Marine biome, the gap between music lasts long
--	--------	--------	---

Obrázek 25 MSUI_TC011 – chyba v ovládání hlasitosti hudby

(zdroj: vlastní)

Při změně kvality textur byla detekována další vada (Obrázek 26). Pokud byla kvalita FSR nastavena v rozmezí od nejnižší po střední, byl zaznamenán nežádoucí efekt vibrování lesního biomu, pravděpodobně způsobený kombinací animace a sníženého rozlišení.

Ensure that each value corresponds to a predefined level of visual quality (e.g. texture display, resolution of distant objects).	Medium	FAILED	For values lowest to Above Medium, it happens that some biomes (e.g. trees) knock unpleasantly.
---	--------	--------	---

Obrázek 26 MSUI_TC023 – chyba v nastavení kvality textur

(zdroj: vlastní)

Další problém (Obrázek 27) byl identifikován v nesprávné změně rozlišení hry po přepnutí do režimu zobrazení v okně. Rozlišení bylo nastaveno na jedno z vybraných hodnot a ke kontrole aktuálních rozměrů okna byl použit skript spuštěný v aplikaci Script Editor (Mac), kterým byla z cílové aplikace načtena a následně zobrazena velikost herního okna.

```
tell application "System Events"
  tell application process "Preserve"
    get size of window 1
  end tell
end tell
```


Verify that the game will not allow longer text to be entered (e.g., trim unnecessary characters).	Medium	FAILED	long texts can be entered
Try entering special characters (e.g. /, \, *, :).	Medium	FAILED	unauthorised characters can be entered
Check that the game warns or disables saving with illegal characters.	Medium	FAILED	nothing is displayed

Obrázek 30 MSUI_TC031 – chyba ve vytváření nové uložené pozice

(zdroj: vlastní)

5.2.7 Puzzle

V testované verzi bylo ověřeno fungování 15 hádanek pro každý biom. Pro každý biom byl založen samostatný excel dokument, v němž každá záložka odpovídala jedné hádance. Testování hádanek proběhlo drtivě pozitivně. V mořském i savana biomu byly všechny hádanky vyhodnoceny bez problémů s připsáním bodů, nadměrnou náročností nebo chybami v UI/UX. Výjimka byla zaznamenána v 12. hádance pro kontinentální biom, kde bylo zjištěno, že dokončení hádanky nebylo možné bez využití bugu umožňujícího nadměrné zavlažení dlaždic a tím i neoprávněné navýšení skóre.

V rámci testování hádanek byla položena otázka, jak objektivně určit a změřit průměrnou náročnost hádanek, aby nebyly chyby v nedokončení přisuzovány pouze individuálním schopnostem hráče. Byla proto definována následující hodnotící kritéria:

- Byla hodnocena *intuitivnost logiky* – zjišťovalo se, zda jsou pravidla pro splnění hádanky jasně definována, aby nejasnost pravidel vysvětlovala neúspěch.
- Byla analyzována *komplexita nápověd* – ověřovalo se, jaký dopad má přítomnost či absence různých druhů nápověd na obtížnost hádanky.
- Byly shromážděny *statistiky hráčů a feedback* – zaznamenávalo se procento hráčů dokončujících hádanku a pomocí dotazníků byly získávány informace o její složitosti a zábavnosti.
- Bylo prováděno *interní testování* – počty chyb a opakující se vzory selhání poskytl data o technické náročnosti hádanky.
- Bylo provedeno *porovnání s existujícími hádkami* – srovnávala se průměrná doba dokončení a počet pokusů u hádanek se stejným principem ve hře či v jiných titulech, aby byla odhadnuta průměrná složitost.

Testování náročnosti a kvality je považováno za složitý proces, v němž dochází k individuálním výsledkům. Proto je doporučeno, aby bylo testováno s více osobami, přičemž jsou analyzovány jejich průběhy, zkoušeny nové kombinace, měněny vstupní parametry a hádanka porovnávána s jinými.

5.2.8 Ovládání

Testování bylo provedeno pomocí testovacího scénáře *TS – Controller*, přičemž bylo otestováno ovládání prostřednictvím ovladače i klávesnice s myší. V rámci scénáře bylo zaměřeno na základní ovládání, přemapování předdefinovaných vstupů (remapping) a konfiguraci ovládání, včetně úprav citlivosti pohybu a převrácení os.

Na první záložce bylo ovládání hry testováno a nebyly zjištěny žádné chyby – ovladač i myš s klávesnicí byly vyhodnoceny jako vyhovující. V remappingu však byla detekována závada (viz Obrázek 31), kdy při přiřazení stejného tlačítka více akcím neměla být volba uložena a mělo být zobrazeno hlášení *Vznikl konflikt v ovládání, prosím změňte nastavení ovládání*. Toto chování bylo správně implementováno v prostředí myši a klávesnice, kde bylo uložení zablokováno a hláška zobrazena, avšak u ovladače bylo zadání více akcí na jedno tlačítko povoleno a nastavení přesto uloženo.

Remap Conflict Detection	Attempt to remap two buttons to the same action (e.g., both A and B buttons to Jump).		PASSED	
	Confirm the remap.		PASSED	
	The game should prevent the conflict, either by showing an error message or automatically suggesting alternative remaps.		FAILED	There is no sign that you used two same buttons on the different things on Controller. On Keyboard is working well.

Obrázek 31 CS_TC004 – Konflikt v přenastavení ovládání

(zdroj: vlastní)

Při testování nastavení ovládání v TC006 (Obrázek 32) mělo být při převrácení funkce přiblížení zajištěno, aby pohyb kolečkem myši dopředu (nebo stisk tlačítka zoom) vedl k přiblížení kamery a pohyb dozadu k jejímu oddálení. Změna chování však nebyla aplikována a ovládání zůstalo v režimu s vypnutým převrácením přiblížení. Tato závada byla zaznamenána jak na ovladači, tak na klávesnici s myší.

Invert Zoom - on	Moving the mouse wheel forward (or pressing the zoom out button) zooms the camera away from the subject.		FAILED	Even when changing the zoom inversion, it behaves the same as if the invert was turned off
	Moving the mouse wheel back (or pressing the zoom out button) brings the camera closer to the subject.		FAILED	

Obrázek 32 TC006 –Nefunkční převrácení zoom

(zdroj: vlastní)

V testovacím případě TC007 (Obrázek 33) bylo zjištěno nesprávné ovládání náklonu. Při nastavení vypnutého převrácení náklonu mělo být stiskem tlačítka nebo kolečka myši a pohybem vpřed dosaženo zvýšení náklonu (směr nahoru) a pohybem vzad jeho snížení (směr dolů). Bez ohledu na provedené nastavení bylo však převrácení náklonu aktivní a nebylo možné jej deaktivovat. Stejné chování bylo zaznamenáno jak na ovladači, tak na klávesnici s myši.

Invert Tilt - off	Press the mouse wheel and move forward (or press the zoom out button) raises the camera (upward tilt).		FAILED	invert tilt is off but joystick up makes movement down and otherwise
	Press the mouse wheel and move back (or press the zoom out button) lowers the camera (downward tilt).		FAILED	

Obrázek 33 TC007 – Nefunkční vypnutí převrácení náklonu

(zdroj: vlastní)

V poslední řadě byly v kombinatorické tabulce odhaleny chyby (pravidla sestavení viz kapitola Kombinatorické testování). Tabulka je zaměřena na testování kombinací různých nastavení a jejich dopadů na hratelnost, přičemž bylo cílem ověřit správnou funkčnost ovládání ve všech možných konfiguracích. Pouze UC3 a UC5 byly vyhodnoceny jako bezchybné. Ve všech ostatních případech se projeví chyby popsané výše. Každý chybný prvek nastavení byl v tabulce označen červenou barvou – v tomto případě bylo převrácení přiblížení nastaveno na ANO a převrácení náklonu na NE.

Combinatorial table								
	Movement speed	Rotation/tilt speed	Controller cursor speed	Invert X camera	Invert Y camera	Invert zoom	Invert tilt	Status
UC1	1	1	1	Yes	Yes	Yes	Yes	FAILED
UC2	5	5	5	No	No	Yes	no	FAILED
UC3	10	10	10	Yes	No	No	yes	PASSED
UC4	1	1	5	No	Yes	No	no	FAILED
UC5	5	5	10	No	Yes	No	yes	PASSED
UC6	10	1	1	No	Yes	No	no	FAILED
UC7	1	1	10	Yes	No	Yes	yes	FAILED
UC8	5	1	10	Yes	No	Yes	no	FAILED
UC9	10	5	1	Yes	No	Yes	yes	FAILED

Obrázek 34 Kombinatorická tabulka ovládání s výsledky

(zdroj: vlastní)

Po vytvoření kombinatorické tabulky (Obrázek 34) a otestování chování bylo zjištěno, že i jediná či dvě chyby mohou způsobit selhání prakticky celé testovací tabulky. V daném případě byla kombinatorická tabulka shledána nevhodnou a nebylo díky ní testování nijak urychleno. Stejných výsledků by bylo dosaženo prostřednictvím klasických UC uvedených výše. Kombinatorická tabulka by dávala smysl až při rozsáhlejších možnostech nastavení či silnější vzájemné závislosti filtrů.

5.3 Lokalizační testování (Localisation testing)

Lokalizační testování bylo zaměřeno na češtinu a angličtinu, přičemž chyby v ostatních jazycích byly zaznamenány pouze okrajově z důvodu vysoké priority a vlivu na hratelnost. Hra byla vyvíjena primárně v angličtině a zbývajících deset jazyků bylo přeloženo strojově. Proto byla pozornost testerů věnována primárně angličtině a dalším jazykům pouze v případě přítomnosti rodilých mluvčích.

Game tree:		Status:
Credits		PASSED
Options	Controls	PASSED
	Graphics	PASSED
	Audio	PASSED
	Language	PASSED
	Other	PASSED
	Erase all progress	PASSED
Compendium	Continental biome	PASSED
	Savana biome	PASSED
	Marine biome	PASSED
	Jurrastic biome	PAUSED
Play	Continental biome <i>Classic game</i>	PASSED

Obrázek 35 Herní strom

(zdroj: vlastní)

V souboru *TS – Localisation* jsou obsaženy čtyři záložky: Main, Terminology, UX UI a Formatting. Záložka Main je koncipována stejně jako ostatní TS v této práci – je v ní uvedena hlavička s popisem a navigační cestou, následně jsou doplněny poznámky k mechanikám a je prezentována tabulka testovacích případů. Tabulka je sestavena z testů zaměřených na přepínání jazyka, kontrolu diakritiky a kontextu, správnost a konzistenci překladů, kulturní vhodnost (např. symboly citlivé v některých kulturách), formátování dat a čísel a UX/UI lokalizační požadavky. Kromě toho je do *TS – Localisation* zařazen

tzv. herní strom, který znázorňuje všechny herní oblasti dostupné hráči a které je třeba při testování lokalizace zohlednit (viz Obrázek 35).

V záložce Terminology je umístěna pomocná tabulka s 11 sloupci pro každou jazykovou mutaci a řádky s anglickými termíny. Obrázek 36 ukazuje část této tabulky. Tabulka je určena jako jednotné místo pro zadávání korektních frází a jednotlivých slov, čímž je zajištěna kontrola konzistentní terminologie napříč celou hrou.

English	Czech	Notes	French	German	European Spanish	Japanese	Simplified Chinese	Italian	Korean	Ukrainian	Slovak
Continental biome	Kontinentální biom										
lowlands	nižiny	Dirrefences in Czech terminology									
highlands	vysočiny	Dirrefences in Czech terminology									
Mountain	hory	Dirrefences in Czech terminology									
Small rain	Malý déšť	Dirrefences in Czech terminology									
Large rain	Velký déšť	Dirrefences in Czech terminology									
Continental River	Kontinentální řeka	Dirrefences in Czech terminology									
Continental Lake	Kontinentální jezero	Dirrefences in Czech terminology									
Flowers	Květiny										
Wetlands	Mokřady										
Tall Grass	Vysoká tráva										
Rock	Kámen	Corectly Skály									

Obrázek 36 Tabulka terminologie

(zdroj: vlastní)

Záložka UX UI je vytvořena se stejným designem jako záložka Main. V hlavní tabulce je uvedeno deset testovacích případů, kterým byla věnována pozornost v oblasti kontrolních prvků hry (např. hlavní menu a nastavení), uživatelského rozhraní v herním prostředí, čitelnosti textu, konzistentnosti terminologie (odkaz na záložku Terminology), grafických prvků a ikon, zpětné vazby na provedené akce, správného zobrazování v různých rozlišeních a na různých zařízeních a zorientování se v herním rozhraní (např. rychlá navigace v menu).

Poslední záložka Formatting je tvořena pomocnou tabulkou obsahující všechny formáty dat a čísel pro jednotlivé jazykové mutace. Obrázek 37 zachycuje výňatek této tabulky, v němž jsou pro každý jazyk uvedeny korektní formy zobrazení data včetně zdroje.

Date		
Language	Correct format	Example
English	MM/DD/YYYY	12/13/24
Czech	DD.MM.YYYY	13.12.2024
French	DD/MM/YYYY	13/12/2024
German	DD.MM.YYYY	13.12.2024
European Spanish	DD/MM/YYYY	13/12/2024
Japanese	YYYY年MM月DD日	2024年12月13日
Simplified Chinese	YYYY/MM/DD	12/13/24
Italian	DD/MM/YYYY	13/12/2024
Korean	YYYY년 MM월 DD일	2024년 12월 13일
Ukrainian	DD.MM.YYYY	13.12.2024
Slovak	DD. MM. YYYY	13. 12. 2024
https://en.wikipedia.org/wiki/List_of_date_formats_by_country		

Obrázek 37 Správné formátování data napříč jazyky

(zdroj: vlastní)

5.3.1 Výstup testování

Veškeré výstupy jsou obsaženy v souboru *Bug report – localisation*. Většina zaznamenaných závad byla nízké priority a spočívala v různých typografických chybách, nesprávných či neúplných překladech a nekonzistentní terminologii. Dále byly identifikovány problémy se zarovnáním textu, nesprávným vykreslením UI elementů a chybějícím zobrazením textací nebo ikon.

V rámci testování partner_build 0.6.70 byly následující testovací případy vyhodnoceny se statusem FAILED:

MAIN:

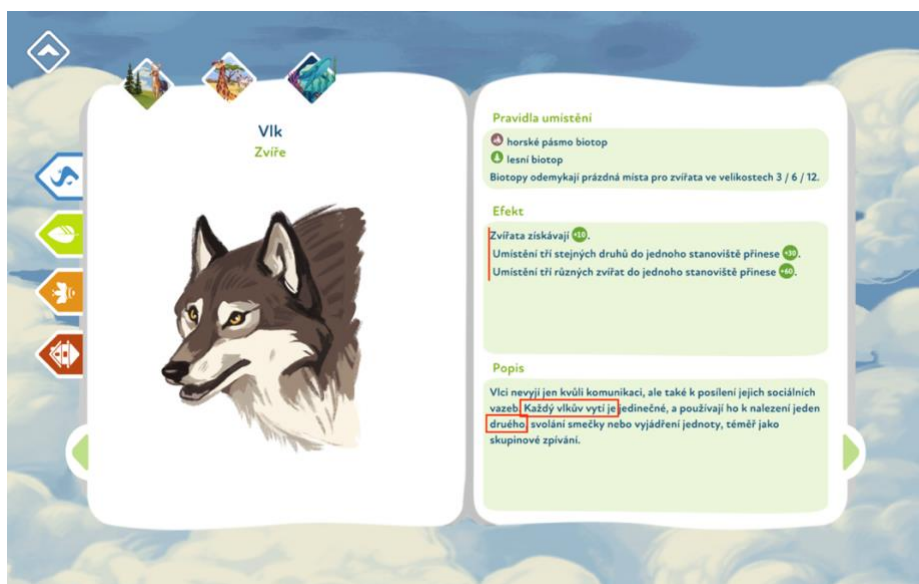
- TC001 – General requirements:
 - Check that no contextual errors (incorrect translation, nonsense text, ...) occur and placeholders are not displayed.
 - Check grammar.
- TC002 – Text translation check:
 - Check that all texts are translated into the set language (use the game tree).
 - Check that there are no grammatical errors or typos in the language.
- TC003 – Consistency of translations:
 - Check that the same terminology is used throughout the game (use the game tree).
- TC004 – Cultural appropriateness:
 - Date, time and unit formatting are adapted to the countries – see Note 3.
 - Check that no characters are missing or replaced by placeholders (e.g. question marks).
- TC005 – Formatting of data and numbers:
 - Check the format of the data for each language (hover text if the mouse is hovered over the continue playing button).
 - Check the number format for each language (Compendium).

- TC006 – UX/UI testing

UX UI:

- UXUI_TC001 – Checking the main menu and game settings:
 - All elements are translated into the chosen language.
 - Translations do not exceed the length of the UI elements (text must not "stick out" from the buttons).
- UXUI_TC002 – Checking the UI in the game environment:
 - Score status and indicators are readable. Icons and text are not overlaid and properly aligned.
 - Text on tabs is localized and legible. Long translations wrap correctly.
- UXUI_TC003 – Text readability and consistency check:
 - Verify that text is not cropped or incorrectly wrapped.
 - Verify that the text is consistent in terminology across the interface.

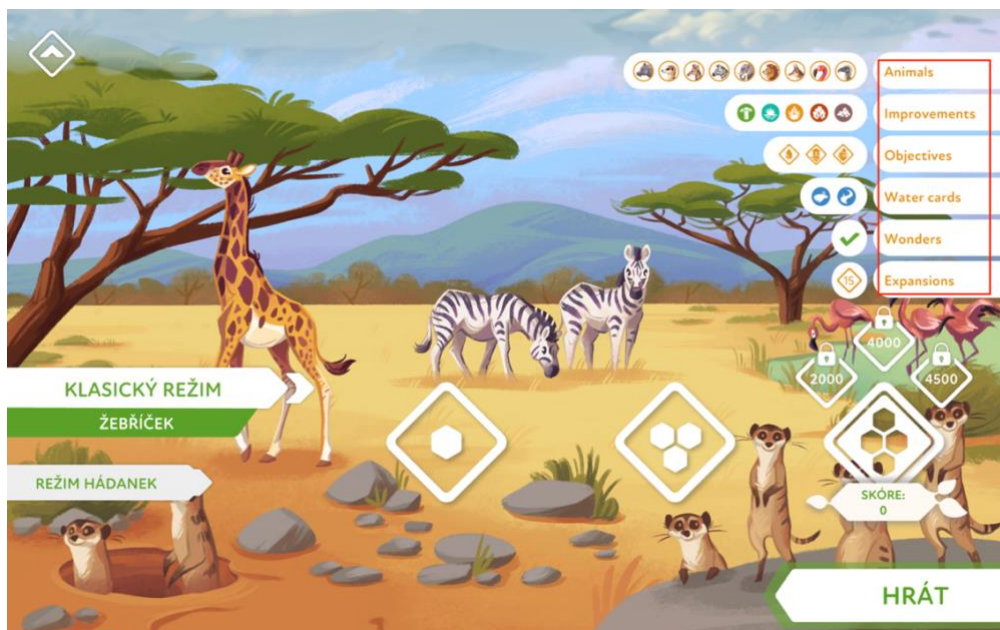
Výstup testování je obvykle převáděn do ticketů v Jiře. Pro účely diplomové práce jsou však bugy zaznamenány v *Bug report – localisation*, kde jsou uvedeny atributy jako ID, kroky k reprodukci, očekávaný a skutečný výsledek, obrazová dokumentace a priorita. Níže jsou uvedeny některé chyby, které byly při testování odhaleny:



Obrázek 38 I. příklad lokalizační chyby

(zdroj: vlastní)

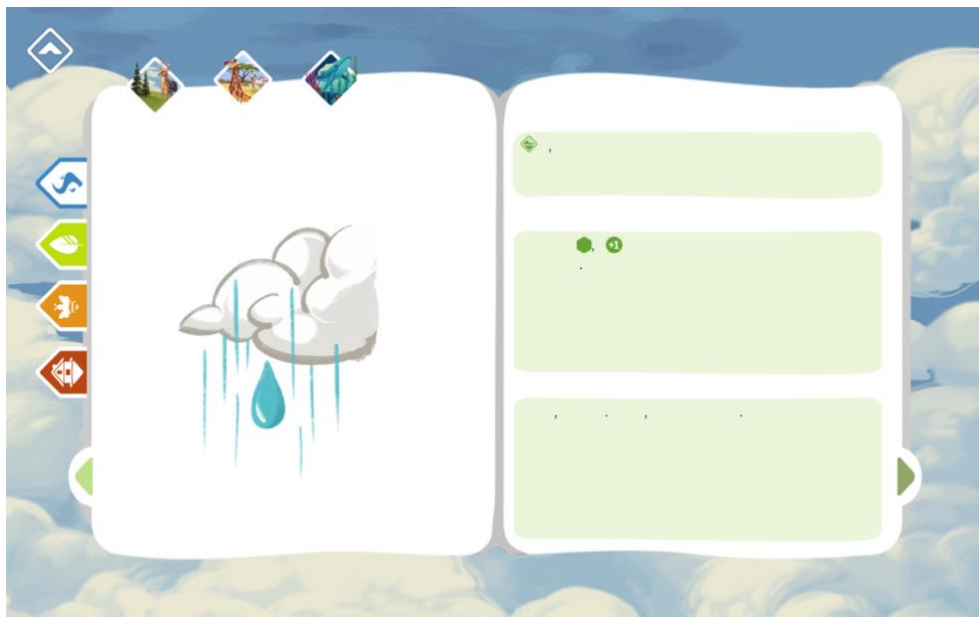
Obrázek 38 je využit k znázornění první nejčastěji se vyskytující skupiny chyb. V Kompendiu jsou zaznamenány často nezarovnané texty v sekci efekty a různé gramatické chyby – chybné pravopisné tvary, nesprávná shoda podmětu s přísudkem či chybné použití pádů. Tyto chyby byly způsobeny strojovým překladem a bylo u nich uvažováno o zapojení komunity k jejich odstranění. K dispozici jsou také nástroje jako Poedit, Grammarly či ChatGPT.



Obrázek 39 II. příklad lokalizační chyby

(zdroj: vlastní)

Obrázek 39 ilustruje případ neprovedených překladů. Infografika před začátkem hraní o vybrané mapě byla v edici 0.6.69 zavedena, avšak byla lokalizována pouze do angličtiny. V aktuální verzi buildu 0.6.70 nebyl překlad doplněn, protože byla upřednostněna oprava závažnějších chyb ohrožujících stabilitu a průchodnost hry.



Obrázek 40 III. příklad lokalizační chyby

(zdroj: vlastní)

Obrázek 40 znázorňuje vyskytující se chyby, a to chybějící překlady v ukrajinštině, korejštině, japonštině a zjednodušené čínštině. V Kompendiu byla použita různá znaková písma (Hanzi pro zjednodušenou čínštinu, Kanji pro japonštinu, Hanja pro korejštinu a ukrajinská cyrilice), v důsledku neimplementování podpory těchto sad však zůstaly rozsáhlé části textu nezobrazeny, a tato závada byla klasifikována jako vysoce prioritní.

Jako nejefektivnější opatření byla doporučena korektura rodilými mluvčími nebo komunita testerů, která by nejprve nahlásila všechny nesrovnalosti. V menších týmech nebo při omezené komunitní podpoře mohou být k vytvoření přesných jazykových verzí využity nástroje typu Poedit, Grammarly či ChatGPT. Doporučeno bylo implementovat jazykové mutace postupně – nejprve u nejrozšířenějších jazyků a po jejich důkladném otestování přecházet k dalším mutacím.

5.4 Testování kompatibility (Compatibility testing)

V rámci testování kompatibility je vyžadováno, aby hra splňovala „pohodlné“ hraní na minimálních i optimálních požadavcích. Ty byly definovány následovně:

Minimální požadavky jsou definovány tak, že hra běží v rozlišení alespoň 720p, plynulost dosahuje 30 FPS, ovládání vykazuje minimální latenci a stabilita je zajištěna bez zásadních problémů. V tomto režimu je hra považována za hratelnou, avšak vizuálně méně atraktivní a s občasnými výpadky plynulosti.

Jako doporučené požadavky je považována konfigurace, v níž je hra zobrazena v rozlišení minimálně 1080p, plynulost snímků je vyšší než 30 FPS (ideálně nad 60), ovládání vykazuje minimální latenci a je zajištěna stabilita bez zásadních problémů. V tomto režimu je hra vyhodnocena jako vysoce hratelná, vizuálně atraktivní a bez výpadků plynulosti.

Ve všech případech musí být zajištěno, aby latence mezi hráčem a hrou nebyla nadměrná, využití paměti zůstalo v přijatelných mezích, doby načítání byly rychlé, zatížení hardwaru nepřekročilo bezpečné limity a aby nedošlo k žádné závadě bránící plynulému hraní.

Nejlepší způsob testování je považován za provedení na skutečném hardwaru. Vzhledem k nemožnosti získat specifické zařízení bylo testování realizováno na systémech s obdobným nebo dostačujícím výkonem. Pro ilustraci byla využita tabulka z PassMark Software (Obrázek 41), ve které jsou zobrazeny benchmarky grafických karet. Z porovnání vyplývá, že Intel UHD 620 dosahuje přibližně pouze 50 % až 33 % výkonu karet uvedených u minimálních HW požadavků (GTX 750 a Radeon HD 7770) na Steamu. Přesto bylo zjištěno, že pro ověření běhu hry na minimálních nastaveních je tento výkon dostatečný, neboť reálné nároky při testu byly nižší než oficiálně uváděné minimum. Intel HD 5000 je do srovnání zařazena pouze pro Linux a Intel Iris 6100 byla použita pro testování Mac verze hry s minimálními požadavky.

	Intel UHD Graphics 620	GeForce GTX 750	Radeon HD 7770	Intel HD 5000	Intel HD 2000	Intel Iris 6100
Price	Search Online	\$135.45 BUY NOW!	\$300 ¹	\$625 ¹	Search Online	Search Online
Bus Interface	Integrated	PCIe 3.0 x16	PCI Express 3.0 x16	NA ²	NA ²	NA ²
GPU Class	Desktop	Desktop	Desktop	NA ²	NA ²	NA ²
Core Clock	300 MHz	1020 MHz	1020 MHz	NA ²	NA ²	NA ²
Memory Clock	1100 MHz	1250 MHz	1150 MHz	NA ²	NA ²	NA ²
Maximum Memory Supported	NA ²	2048 MB	2048 MB	NA ²	NA ²	NA ²
DirectX	12.0	12.0	11.2	NA ²	NA ²	NA ²
OpenGL	4.4	4.5	4.2	NA ²	NA ²	NA ²
Max TDP	15W	55W	80W	NA ²	NA ²	NA ²
First Seen on Chart	Q2 2017	Q1 2014	Q2 2012	Q2 2013	Q2 2015	Q1 2015
# of Samples	25511	6479	9359	1147	1	600
Average Results (% diff. to max in group)						
DirectX 9	21 FPS (-62.9%)	57 FPS	41 FPS (-28.0%)	9 FPS (-83.6%)	NA	15 FPS (-74.4%)
DirectX 10	4 FPS (-76.4%)	16 FPS	8 FPS (-52.2%)	2 FPS (-86.2%)	NA	3 FPS (-79.1%)
DirectX 11	8 FPS (-62.9%)	22 FPS	13 FPS (-40.8%)	4 FPS (-82.6%)	NA	6 FPS (-75.4%)
DirectX 12	5 FPS (-67.0%)	14 FPS	12 FPS (-15.1%)	3 FPS (-81.0%)	NA	5 FPS (-66.9%)
GPU Compute	1051 Ops/Sec (-26.8%)	1435 Ops/Sec	1157 Ops/Sec (-19.4%)	448 Ops/Sec (-68.8%)	NA	533 Ops/Sec (-62.9%)
G2D Rating	242 (-52.7%)	511	461 (-9.7%)	221 (-56.7%)	456 (-10.8%)	314 (-38.6%)
G3D Rating	1043 (-68.8%)	3342	2171 (-35.0%)	584 (-82.5%)	213 (-93.6%)	903 (-73.0%)

Obrázek 41 Výkonnostní porovnání GPU z tabulek a vlastněných autorem

(zdroj: PassMark Software [73])

5.4.1 Windows

Tabulka 2 prezentuje minimální a doporučené parametry pro Windows. Rozdíly jsou zaznamenány ve velikosti paměti RAM, použitých grafických kartách a výkonu procesoru. Procesor není v tabulce přímo specifikován, ale je uveden pouze počet jader a takt. Testování v rámci doporučených parametrů bylo realizováno v kapitole výkonostního testování na zařízení ROG Ally.

V případě minimálních požadavků nebyla některá nastavení v souladu s uvedenou tabulkou. Testování bylo provedeno na operačním systému Windows 11, jako GPU byla využit integrovaná Intel UHD Graphics 620 a vzhledem k absenci přesného zápisu procesoru byl pro testování použit Intel Core i7-8550U (1,8 GHz, 4 jádra) (viz Obrázek 42).

Tabulka 2 HW a SW požadavky Windows

	Minimální:	Doporučené:
OS:	Windows 10	Windows 10
CPU:	2 GHz Dual Core (Intel / AMD)	2.5 GHz Dual Core (Intel / AMD)
RAM:	8 GB RAM	16 GB RAM
GPU:	Nvidia GeForce GTX 750 / AMD Radeon HD 7770	Nvidia GeForce GTX 960 / AMD Radeon RX560
HDD:	1 GB volného místa	1 GB volného místa

(zdroj: Steam [70][73])

Obrázek 42 je záznamem obrazovky z DirectX rozhraní, kde je detailně popsán veškerý HW v počítači.



Obrázek 42 Minimální požadavky – HW – Windows

(zdroj: vlastní)

Bylo testováno po dobu minimálně 30 minut. Během testu byly ve hře monitorovány hodnoty FPS a využití paměti, zatímco v rámci Správce úloh bylo sledováno zatížení CPU, GPU a RAM. Zohledněna byla jak plynulost běhu, tak vytížení systémových prostředků. Hra byla spuštěna s nejnižšími detaily, rozlišením 1024x768px a s vypnutým globálním osvětlením, ambientní okluzí a stíny. Kromě hry byly aktivní pouze aplikace Steam a Správce úloh. Byla otestována volná hra pro všechny tři biomy.

Pro všechna tři testovaná prostředí byla udržována stabilní snímková frekvence s průměrem 48 FPS, minimem 27 FPS a maximem 57 FPS. Z hlediska výkonu byla hra

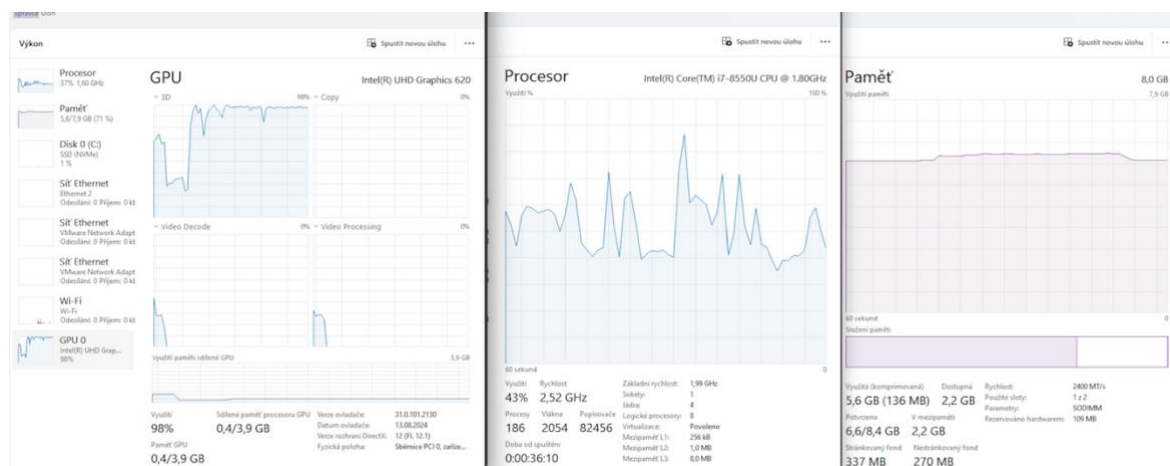
považována za plynule hratelnou i na slabší HW sestavě. Grafické artefakty se však objevovaly při nižším nastavení kvality textur, kdy docházelo k nežádoucímu rozmazávání objektů a třepotání prvků. Obrázek 43 ukazuje rozmazané textury habitatů, neostře texty a ikony a blikající černobílé artefakty na pozadí.



Obrázek 43 Grafické glitche a artefakty při nízkém rozlišení

(zdroj: vlastní)

Obrázek 44 znázorňuje vytížení systému během hraní. Využití GPU bylo udržováno na 98 %, RAM byla využita ze 72 % a CPU bylo zatíženo průměrně na 43 %. Ze záznamu je patrné, že nejvíce zátěže bylo kladeno na GPU, které bylo zodpovědné za kvalitu textur a plynulost snímkování. Právě toto zatížení vedlo k průměrné snímkové frekvenci 48 FPS, která však místy klesla až na 27 FPS.



Obrázek 44 Vytíženost PC

(zdroj: vlastní)

5.4.2 Mac

Tabulka 3 opět uvedla minimální a doporučené HW požadavky. Pro ověření minimální konfigurace byl použit MacBook Pro z roku 2015 s operačním systémem macOS 12.6.6 Monterey, procesorem Intel Core i5 (2,7 GHz, 2 jádra) a integrovanou grafikou Intel Iris Graphics 6100, jejíž výkon je oproti doporučené grafice nižší přibližně o ~270 %.

Tabulka 3 HW a SW požadavky Mac

	Minimální:	Doporučené:
OS:	MacOS 11 or later	MacOS 11 or later
CPU:	Intel Core i3 2.00 GHz or AMD equivalent	Intel Core i5 3.00GHz or AMD equivalent
RAM:	8 GB RAM	16 GB RAM
GPU:	Nvidia GeForce GTX 750 or higher	Nvidia GeForce GTX960 or higher
HDD:	1 GB volného místa	1 GB volného místa

(zdroj: Steam [70])

Obrázek 45 zobrazuje detailní hardware MacBooku získaný ze systémových informací. Testování bylo navrženo tak, aby proběhlo za stejných podmínek jako na Windows, nicméně bylo zjištěno, že při načítání herní mapy došlo po přibližně 15 sekundách k zamrznutí animace zvířete, které přetrvalo dalších 15–20 sekund. Následně byla detekována blikající černá obrazovka a proces načítání se již neobnovil (nahrávka je dostupná v příloze diplomové práce). Tento jev byl zaznamenán ve všech herních režimech. Při spuštění na novějším MacBooku s čipem M1 bylo načítání mapy dokončeno bez problémů a byla potvrzena plná funkčnost. Jako pravděpodobné příčina byla vyhodnocena nekompatibilita buildu se staršími Mac čipy nebo změny v grafickém API Unity, které způsobily selhání spuštění.

Přehled hardwaru:		Intel Iris Graphics 6100:	
Název modelu:	MacBook Pro	Model čipové sady:	Intel Iris Graphics 6100
Identifikátor modelu:	MacBookPro12,1	Typ:	GPU
Název procesoru:	Dvoujádrový Intel Core i5	Sběrnice:	Vestavěný
Rychlost procesoru:	2,7 GHz	VRAM (dynamicky, max.):	1536 MB
Počet procesorů:	1	Dodavatel:	Intel
Celkový počet jader:	2	ID zařízení:	0x162b
L2 mezipaměť (na jádro):	256 kB	ID revize:	0x0009
L3 mezipaměť:	3 MB	Řada Metal:	Podporováno, řada GPU Metal, macOS 1
Technologie Hyper-Threading:	Zapnuto	Monitory:	
Paměť:	8 GB	Barevný LCD:	
Verze systémového firmwaru:	481.0.0.0.0	Typ monitoru:	Vestavěný Retina LCD
Verze zavaděče operačního systému:	540.120.3~22	Rozlišení:	2560 x 1600 Retina
Verze SMC (systém):	2.28f7	Hloubka mezipaměti snímků:	24bitové barvy (ARGB8888)
Sériové číslo (systém):	C02S68ZYFVH5	Hlavní monitor:	Ano
UUID hardwaru:	C5260431-1AFF-5B91-95E9-D79BDF4CC629	Zrcadlení:	Vypnuto
UDID pro zřizování:	C5260431-1AFF-5B91-95E9-D79BDF4CC629	Online:	Ano
		Automatické nastavení jasu:	Ne
		Typ připojení:	Interní

Obrázek 45 Minimální požadavky – Mac

(zdroj: vlastní)

V rámci testování doporučené konfigurace byl využit MacBook Air z roku 2020, který byl vstupním modelem Applu s čipem Apple M1 a mohl tak reprezentovat skupinu uživatelů vlastníci tento či novější čip. Zařízení bylo vybaveno 16 GB RAM, čtyřmi výkonnými a čtyřmi úspornými jádry CPU a integrovanou GPU (viz Obrázek 46).

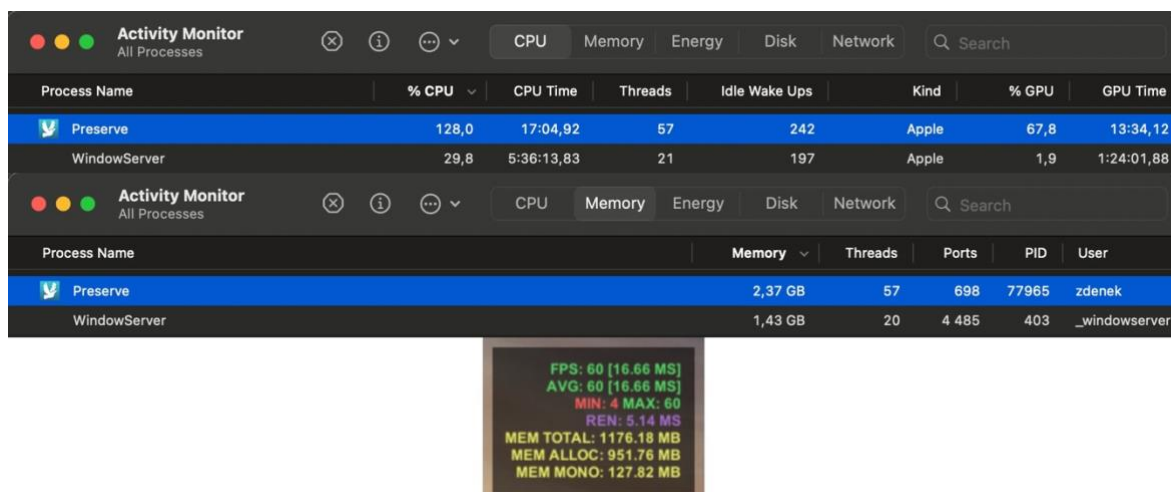
Testování probíhalo minimálně 30 minut, přičemž bylo nepřetržitě monitorováno zobrazení FPS a využití paměti ve hře a v systému byly zaznamenávány metriky využití CPU, GPU a RAM. Hra byla spuštěna s nadprůměrnými detaily, rozlišením 1680x1050px a aktivovaným globálním osvětlením, ambientní okluzí a stíny. Kromě hry byla aktivní pouze aplikace Steam a Správce úloh, přičemž za celou dobu testu byla ověřena stabilita a plynulost volné hry ve všech třech biomech.

Hardware Overview:		Apple M1:	
Model Name:	MacBook Air	Chipset Model:	Apple M1
Model Identifier:	MacBookAir10,1	Type:	GPU
Model Number:	Z12800053CZ/A	Bus:	Built-In
Chip:	Apple M1	Total Number of Cores:	8
Total Number of Cores:	8 (4 performance and 4 efficiency)	Vendor:	Apple (0x106b)
Memory:	16 GB	Metal Support:	Metal 3
System Firmware Version:	11881.81.4	Displays:	
OS Loader Version:	11881.81.4	Color LCD:	
Serial Number (system):	C02GT0TYQ6LX	Display Type:	Built-In Retina LCD
Hardware UUID:	8E126B0D-3558-59C8-86BE-7CB46E25664A	Resolution:	2560 x 1600 Retina
Provisioning UDID:	00008103-001165D62E39001E	Main Display:	Yes
Activation Lock Status:	Enabled	Mirror:	Off
		Online:	Yes
		Automatically Adjust Brightness:	No
		Connection Type:	Internal

Obrázek 46 Doporučené požadavky – Mac

(zdroj: vlastní)

Všemi třemi testovanými zařízeními byla udržena stabilní snímková frekvence s průměrem 60 FPS, minimem 4 FPS a maximem 60 FPS. Z hlediska výkonnosti byla hra vyhodnocena jako plynule hratelná, a bylo konstatováno, že uživatel se střední HW sestavou nenarazí na problémy. Grafické glitche a artefakty zaznamenané při nižším rozlišení se při středním nastavení neprojeví. Bylo zaznamenáno průměrné využití paměti RAM přibližně 2,40 GB. Vytížení CPU a GPU bylo vyhodnoceno na cca 128 % a 68 %, což poukazuje na méně optimalizovaný rendering a správu paměti. Při delším testu bylo dále zjištěno zvýšení teploty notebooku v důsledku vyššího zatížení CPU a GPU (viz Obrázek 47).



Obrázek 47 Vytíženost Mac

(zdroj: vlastní)

5.4.3 Steam/Linux

Tabulka 4 uvedla minimální HW požadavky pro OS Linux. Pro testování byl využit hardware s OS Ubuntu 24.04.1 LTS, procesorem Intel Pentium B960x2, integrovanou grafikou Intel HD Graphics 2000 a 6 GB RAM. Kromě procesoru, který nebyl v požadavcích specifikován, byl zjištěn výkonový rozdíl u GPU. Ta byla oproti požadované slabší přibližně o ~174 %, zatímco kapacita RAM byla o 2 GB vyšší (viz Obrázek 41 s porovnáním jednotlivých GPU). Obrázek 48 prezentoval předpokládané výsledky testovaného hardwaru.

Tabulka 4 Steam/Linux

	Minimální:
OS:	Most recent version of Ubuntu LTS
CPU:	2.0 GHz Intel
RAM:	4 GB RAM
GPU:	Intel HD Graphics 5000 (AMD or NVIDIA equivalent)
HDD:	1 GB volného místa

(zdroj: Steam [70])

Pro test byly stejné podmínky jako při testu verze pro Windows (doba trvání, sledování parametrů a podmínky testu). Během testu byla vyzkoušena volná hra (pro všechny 3 biomy) a hra měla nastaveny nejnižší detaily, rozlišení 1024x768px a vypnuté globální osvětlení, ambientní okluzi a stíny.



Obrázek 48 Vytíženost Linux

(zdroj: vlastní)

Ve všech případech byla snímková frekvence udržována v průměru na 4 FPS. Na takto nízkou snímkovou frekvenci bylo prokázáno, že měl zásadní vliv procesor s integrovanou grafickou kartou, která se ukázala být příliš slabá. I při snížení rozlišení na 640x480p byly FPS zvýšeny pouze na 7 FPS místo 4. Přitom bylo CPU zatíženo na přibližně 40 % a RAM byla využita v objemu 4 GB.

Bylo zjištěno, že při spuštění hry na notebooku HP ProBook 450 G5 s Ubuntu 24.04.1 LTS byla dosažena výrazně lepší snímková frekvence. Hra byla nastavena na střední detaily, rozlišení 1280x800px a bylo aktivováno globální osvětlení, ambientní okluze a stíny. S aplikací běžely pouze aplikace Steam a Správce úloh. Volná hra byla otestována ve všech třech biomech. V úvodní fázi, kdy herní svět obsahoval dvě plošiny, se snímková frekvence pohybovala mezi 45 a 50 FPS. Po přibližně dvaceti minutách a rozšíření mapy o deset segmentů klesla na průměrných 33 FPS.

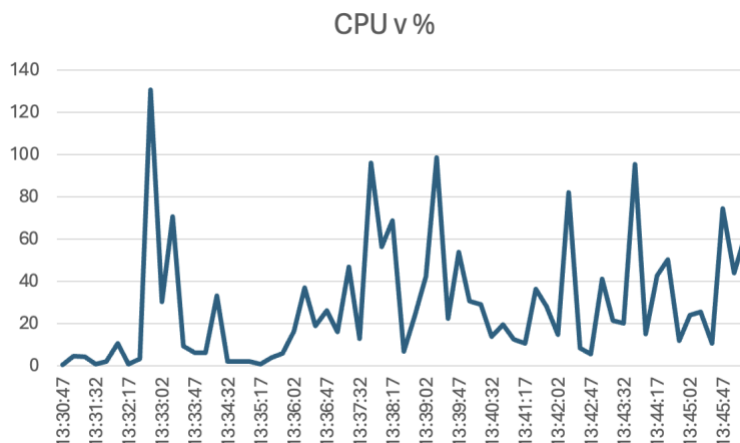
Po přepnutí na nejnižší detaily (ostatní nastavení zůstalo beze změny) byla snímková frekvence na počátku stabilně 60 FPS a po deseti rozšířeních se ustálila na 43 FPS, což představuje úbytek přibližně 17–20 FPS.

Byly zaznamenány výsledky dvou následujících grafů v Ubuntu, přičemž jejich podoba zůstává shodná na všech testovaných operačních systémech. Využitím nástroje Pidstat bylo sledováno zatížení hardwaru a naměřená data byla následně převedena do tabulky a grafu. Prvním příkazem bylo zajištěno měření procesu Preserve.x86_64 každých 15 sekund s parametry pro záhlaví, CPU, paměť a I/O. Druhým příkazem byl log převeden do formátu CSV.

```
pidstat -h -u -r -d -C Preserve.x86_64 15 > pidstat_Preserve.log
```

```
tail -n +4 pidstat_Preserve.log | sed 's/[[:space:]]\+/,/g' > pidstat_Preserve.csv
```

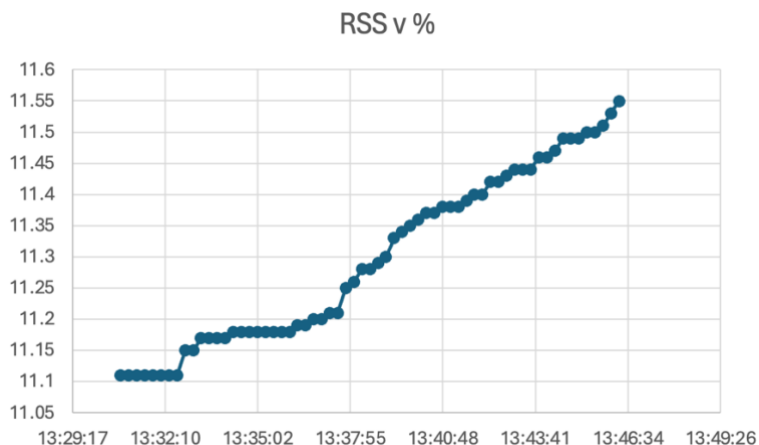
Obrázek 49 znázorňuje procentuální zatížení CPU. V období 13:32–13:33 bylo zaznamenáno nárůst nad 120 %, což bylo způsobeno využitím multithreadingu při zakládání nové hry. Využití v rozmezí 80 až 100 % bylo pozorováno při přidávání většího počtu zvířat či biomů v krátkém časovém úseku. Průměrné zatížení se pohybovalo mezi 20–60 %, což odráží dynamický průběh hry s cyklickými výpočtovými fázemi, kdy se náročné výpočty střídají s kratšími pauzami či I/O operacemi.



Obrázek 49 Využití CPU na střední detaily

(zdroj: vlastní)

Ve druhém grafu (Obrázek 50) je zobrazován podíl fyzické paměti procesu na RAM vyjádřený v procentech během stejného časového úseku. Bylo zaznamenáno trvalé zvyšování využití paměti z 11,10 % na 11,55 %, což svědčí o postupné navyšující se alokaci a ponechávání datových struktur bez výrazného uvolňování. Byl patrný průběžný progres, kdy s rozšiřováním herního světa docházelo k postupnému nárůstu používané paměti.



Obrázek 50 Využití paměti na střední detaily

(zdroj: vlastní)

I když nemohlo být testování provedeno na přímo stanoveném minimálním hardwaru a bylo využito zařízení HP (s GPU s přibližně o 78 % vyšším výkonem). Tak na něm bylo zjištěno, že hra je při minimálních i středních detailech přijatelně hratelná. V žádné z verzí pro macOS či Windows nebyly zaznamenány žádné abnormality.

5.5 Výkonnostní testování (Performance testing)

Bylo ověřeno efektivní využití systémových zdrojů a chování hry v reálných i krajních podmínkách. Byly zohledněny aspekty jako spotřeba baterie na přenosných zařízeních, chování hry při dlouhodobém provozu a jeho dopad na systémové zdroje, stejně jako reakce na nečekaná přerušení (ztráta připojení, usnutí). U testu kompatibility bylo primárně kladeno zaměření na ověření kompatibility se systémem, přičemž měření FPS a zatížení bylo využito výhradně k potvrzení této kompatibility.

5.5.1 Testování spotřeby baterie a vykreslovacího výkonu (Battery Consumption & Rendering Performance Testing)

Cílem testu bylo odhalit úroveň optimalizace hry v reálném provozu napříč různými herními režimy a případné nedostatky. Předpokladem bylo nabití zařízení na 100 %, jeho odpojení od napájení při spuštění hry a sledování výkonu prostřednictvím FPS, spotřeby ve Wattech a procenta zbývajících nabití. Během testu byla aktivní pouze testovaná aplikace, přičemž síťové připojení zůstalo v provozu. Za úspěšné bylo považováno udržení stabilní snímkové frekvence a konstantní spotřeby při daném nastavení, spolu s výdrží zařízení minimálně 3 hodiny.

Pro srovnání byly vybrány dva další tituly s dlouhodobou produkční historií a převážně údržbovým vývojem: akční RPG Kingdom Come: Deliverance od studia Warhorse jako zástupce AAA her s otevřeným světem a strategická tahová hra Terraforming Mars, vybraná kvůli žánrové příbuznosti a podobné hratelnosti.

5.5.1.1 Režim výkonu

Pro tento režim bylo zvoleno nastavení charakterizující optimální rozlišení a snímkovou frekvenci, aby byl hráči zajištěn nejlepší možný herní zážitek. Na handheldu byl aktivován režim Výkon s limitem spotřeby 15 W, byla povolena technologie AMD RIS pro vyostření rozmazaných hran a bylo zvýšeno zrychlení CPU, které umožňuje dočasné navýšení frekvence procesoru (u Z1 Extreme z 3,3 GHz na 5,1 GHz). Displej byl nastaven na 1080p při 120 Hz. Během testu všech titulů se spotřeba pohybovala mezi 23 a 26 W.

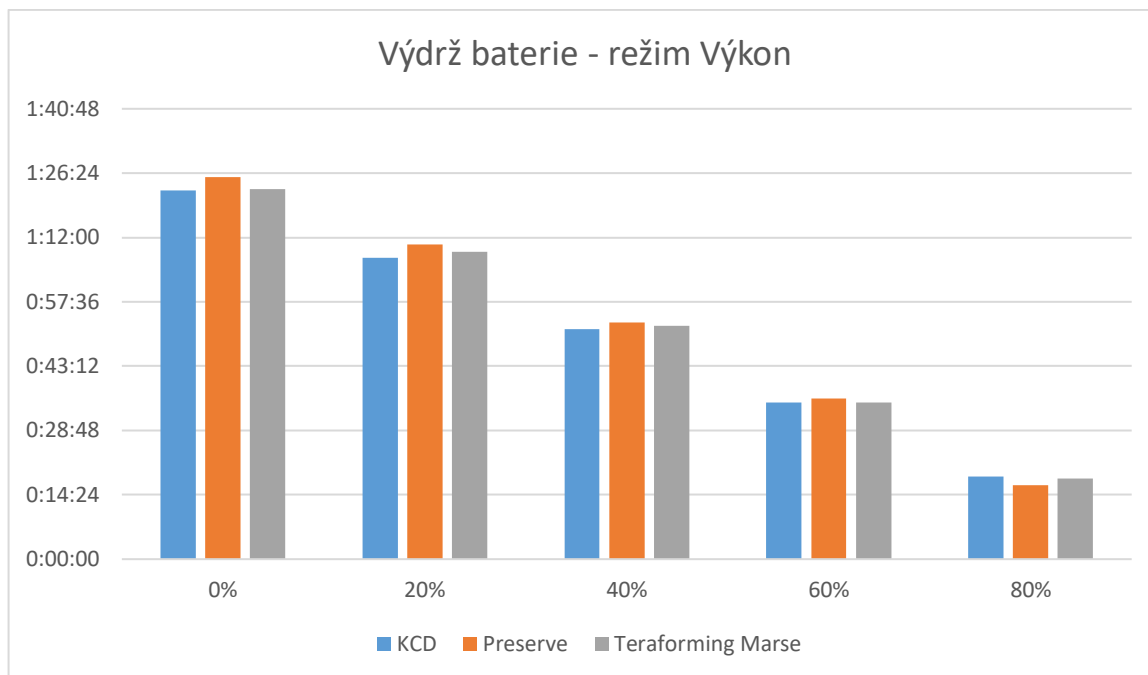
U hry Kingdom Come: Deliverence (zkráceně KCD) bylo testování provedeno hraním hlavní dějové linie, přehráváním cut-scén a volným procházením mapy. Nastavení bylo zvoleno na vysoké detaily a průměrná snímková frekvence se pohybovala mezi 25 a 30 FPS. V případě Preserve byla realizována klasická hra na střední úrovni pro biomy kontinentální a savana a dále byly otestovány vybrané hádanky s nastavením ultra vysoké, při kterém dosahovalo FPS hodnot mezi 70 a 120. U hry Terraforming Mars (zkráceně TM) byl absolvován kompletní tutoriál a Solo challenge. Nastavení detailů bylo na vysoké a snímková frekvence byla stabilizována kolem 80 FPS.

Tabulka 5 Vytížení při režimu Výkon

	80 %	60 %	40 %	20 %	0 %
KCD	0:18:30	0:35:00	0:51:30	01:07:30	01:22:30
Preserve	0:16:30	0:36:00	0:53:00	1:10:30	1:25:30
TM	0:18:00	0:35:00	0:52:10	1:08:45	1:22:45

(zdroj: vlastní)

Z tabulky a obrázku (Tabulka 5 a Obrázek 51) je patrné, že všechny testované hry dosáhly prakticky totožných dob výdrže baterie. V počáteční fázi byla u hry Preserve zaznamenána rychlejší spotřeba, avšak od úrovně 60 % se trend obrátil a byla překonána výdrž ostatních titulů o tři minuty. Přestože je Preserve stále v předběžném přístupu, bylo prokázáno, že je plně hratelné na výkonnějších zařízeních s adekvátním chlazením. Z testu dále vyplynulo, že s rozrůstajícím se herním světem dochází k poklesu FPS ze 120 (omezujících maximální obnovovací frekvenci displeje) na hranici 70 snímků a k nárůstu využití paměti. V pozdějších fázích testování mohla snímková frekvence klesnout i pod 30 FPS, což naznačuje škálovací charakter problému při práci s rozsáhlými mapami (tento problém byl odhalen i v rámci jiných testů). Doporučuje se proto zaměřit optimalizační úsilí na výkonové aspekty spojené s velkými herními oblastmi.



Obrázek 51 Graf výdrže baterie – Výkon

(zdroj: vlastní)

5.5.1.2 Režim výdrže

Pro tento režim bylo zvoleno nastavení charakterizující optimální poměr výdrže baterie a kvality obrazu: na handheldu byl aktivován Tichý režim s limitní spotřebou 10 W, obnovovací frekvence byla snížena na 60 Hz a rozlišení na 1600x900px. Technologie AMD RIS i zrychlení CPU byly v tomto režimu deaktivovány.

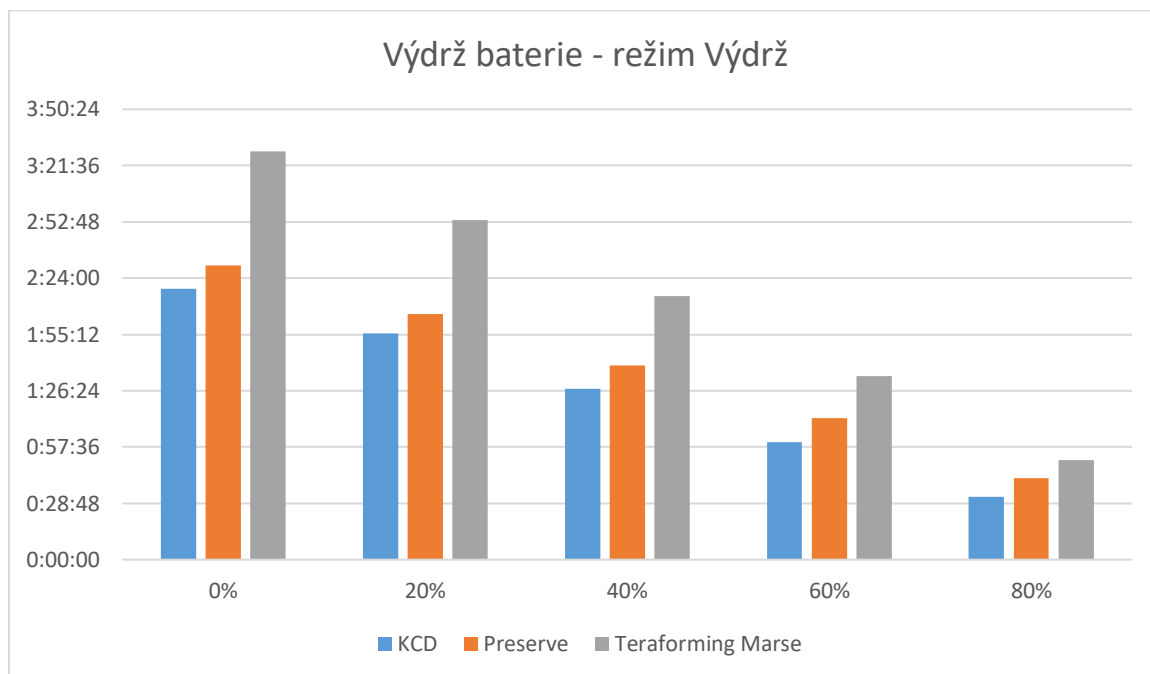
Hra Kingdom Come: Deliverance byla spuštěna na nízké detaily a během měření byla spotřeba udržována mezi 14 a 15 W, přičemž tento odběr se neměnil ani ve městě, ani v přírodě. Snímková frekvence se pohybovala mezi 25 a 30 FPS. Hra Preserve byla testována na střední a nejvyšší úrovni pro mořský a savana biom. Bylo vybráno nejnižší grafické nastavení. Snímková frekvence byla fixována na 60 FPS, ale s rozrůstáním mapy a počtu biomů vzrostla spotřeba až na 15 W (z počátečních 8 W v menu). U hry Terraforming Mars proběhly lokální zápasy proti umělé inteligenci při nízkých detailech, kdy byla spotřeba stabilní na 9–10 W a snímková frekvence setrvala na 60 FPS.

Tabulka 6 Vytížení při režimu Výdrž

	80 %	60 %	40 %	20 %	0 %
KCD	0:32:00	1:00:00	1:27:30	01:55:35	02:18:45
Preserve	0:41:50	1:12:30	1:39:30	2:05:45	2:30:30
TM	0:51:00	1:33:45	2:14:40	2:53:40	3:29:00

(zdroj: vlastní)

Podrobné informace o výdrži byly opět prezentovány v grafu a tabulce (Obrázek 52 a Tabulka 6). Z těchto podkladů je zřejmé, že při optimalizaci bylo dosaženo snížení spotřeby na 10 W, díky čemuž mohl handheld vydržet téměř 3,5 hodiny. Spotřeba her Preserve a Kingdom Come: Deliverance se v průběhu testu přibližovala, přičemž u KCD byla díky otevřenému světu a vyšší výpočetní náročnosti potvrzena spotřeba až 15 W. Hlavním zjištěním bylo, že spotřeba zůstala po celou dobu konstantní. U hry Preserve byla vyhodnocena potřeba zaměřit se na další optimalizaci vykreslování světa v reálném čase a zpracování textur a animací, zvláště s ohledem na trend nízko spotřebních architektur CPU a GPU.



Obrázek 52 Graf výdrže baterie – Výdrž

(zdroj: vlastní)

5.5.2 Testování vytrvalosti (Endurance testing)

Tento test je zaměřen na ověření stability při dlouhodobém nepřerušovaném běhu. Zároveň je považováno za důležité potvrzení, že hra netrpí špatnou správou paměti, přetížením procesoru ani jiným negativním ovlivněním výkonu.

Byla zajištěna nepřetržitá hra po dobu 10 hodin, během níž byl ve Správci úloh neustále sledován výkon procesoru, využití RAM, úložiště a síťová aktivita. Současně byla ve hře monitorována snímková frekvence a využití paměti. Testování bylo provedeno na platformě Windows s nastavením kvality textur „Nadprůměrná“, rozlišením 1280x800px a bez dalších grafických efektů. Hra byla spuštěna v okně, přičemž během prvních 20 minut byl vedle herního okna otevřen Správce úloh. Po uplynutí této doby bylo testování přerušeno, přičemž každou hodinu byly pořizovány snímky obrazovky pro dokumentaci aktuálního zatížení HW a stavu FPS a paměti. Zhruba každé dvě hodiny byl na hře proveden progres, který trval cca 20 minut, aby byly ověřeny postup a funkčnost ovládání a herních prvků. Po deseti hodinách byl herní proces ponechán v klidu po další hodinu, po jejímž uplynutí byly opětovně otestovány všechny herní mechaniky, hra byla následně ukončena a byly vyhodnoceny výsledky testu.

Byl vytvořen krátký testovací scénář (*TS – Performance*), ve kterém byly ve třech testovacích případech ověřeny chování při dlouhodobém testování, kontrola audiovizuálních prvků a kontrola ovládání. Po spuštění a odehrání prvních 20 minut byly zaznamenány následující hodnoty:

- Průměrné FPS: 33 (průměrná doba vykreslení jednoho snímku byla 30,30 ms)
- Celkové využití paměti měřené ve hře (alokovaná + mono): 820,78 MB
- CPU: 21,7 %
- Využití RAM ve Správci úloh: 1509,7 MB

Po uplynutí tří hodin byly zaznamenány minimální změny. Průměrná snímková frekvence byla snížena o tři FPS (s nárůstem průměrné doby vykreslení snímku o tři ms), využití paměti bylo navýšeno o 12 MB a průměrné hodnoty CPU a RAM dosáhly 27 % a 1 565 MB. Během tohoto období bylo zaznamenáno rozšíření mapy a přidání nových biomů, což vysvětluje růst využití paměti a mírné snížení FPS (tato závada je již uvedena v podkapitolách Režim výkonu a Režim výdrže). Všechny naměřené hodnoty byly považovány za přijatelné.

Po uplynutí šesti hodin byla mapa upravena tak, aby odpovídala velikosti během předchozích měření. Průměrný počet snímků byl 33 FPS (s průměrnou dobou vykreslení jednoho snímku 30,30 ms). Využití paměti dosáhlo 884,78 MB. Průměrné zatížení CPU činilo 21,5 % a využití RAM bylo 1 727,7 MB. Kromě navýšení alokované paměti o přibližně 60 MB a zvýšení využití RAM o zhruba 200 MB nedošlo k žádným zásadním odchylkám. Hratelnost, grafika, ovládání i zvuková složka byly vyhodnoceny jako bezproblémové.

Po deseti hodinách nebyly zjištěny žádné zásadní problémy a hra byla vyhodnocena jako bezproblémově hratelná. Na závěr testu byly zaznamenány tyto hodnoty:

- Průměrné FPS: 32 (s průměrnou dobou vykreslení jednoho snímku 31,25 ms)
- Celkové využití paměti měřené ve hře (alokovaná + mono): 884,78 MB
- CPU: 22,5 %
- Využití RAM ve správci úloh: 1734 MB

Z naměřených dat vyplývá, že hra netrpí úniky paměti, přetížením procesoru ani jinými negativními dopady na výkon. Stabilita byla potvrzena i po dlouhodobém běhu a v režimu nečinnosti – vizuální chyby, chybějící zvuk či zvýšená latence ovládání nebyly pozorovány.

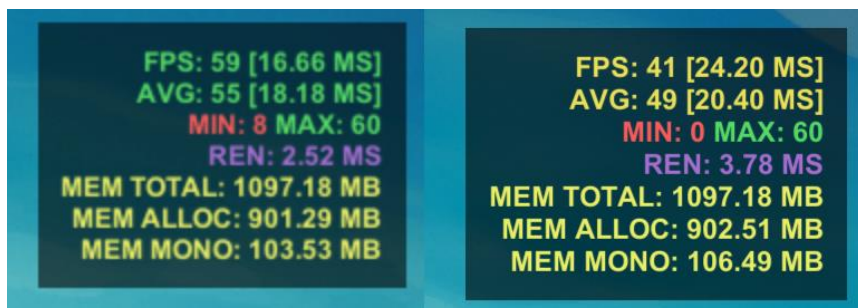
5.5.3 Testování přerušení (Interrupt testing)

Mělo by být ověřeno, že při přerušení hry během hraní (např. usmáním počítače nebo minimalizací okna) nevznikne žádný problém pro hráče, který by mohl zkazit zážitek, a v případě jeho výskytu je závada zaznamenána a odeslána k opravě. K tomu jsou měřeny pocitová latence, FPS a zatížení HW.

Byl sestaven testovací scénář (*TS – Performance*) složený z 11 testovacích případů. Byly v něm zahrnuty případy přepínání aplikace (např. přes Alt + Tab), minimalizace a maximalizace okna, přepínání mezi celoobrazovkovým a okenním režimem, příchozí notifikace, změny napájení, připojení a odpojení periferií (sluchátka, ovladače, klávesnice/myš), ztráta a obnovení internetového připojení, usnutí a probuzení systému, nahrávání obrazovky a přepnutí uživatele.

Testování bylo provedeno na všech třech platformách a výsledky byly vyhodnoceny jako uspokojivé. K problému došlo pouze v jednom testovacím případě, kdy po zmenšení okna došlo k až třetinovému propadu snímkové frekvence, která se ani po obnovení původní

velikosti okna nevrátila na předchozí hodnoty. Obrázek 53 zobrazuje tento jev – na levé straně je zachycen stav před změnou (57–60 FPS), zatímco po změně byly snímky v rozsahu 40–43 FPS.



Obrázek 53 Porovnání FPS před a po změně velikosti okna hry

(zdroj: vlastní)

Jako jedna z možných příčin byla identifikována funkce herního režimu dostupného na macOS, která při nastavené plné obrazovce umožňuje proces hry upřednostnit na nejvyšší úroveň a tím zajistit maximální výkon CPU a GPU. I po deaktivaci tohoto režimu byl problém stále pozorován, a proto byl závěr učiněn, že se jedná o chybu v optimalizaci.

5.6 Testování dodržování předpisů (Compliance testing)

Hra byla vydána na platformě Steam, která při registraci a kontrole aplikace zajišťuje většinu potřebných pravidel a předpisů. Přístup do Steamworks, kde jsou spravována nastavení verzí a buildů, achievementů či slev, nebyl poskytnut, a proto byla tato část považována za otestovanou již samotným vystavením hry na platformu. Testování compliance se tak omezilo na menší rozsah a bylo soustředěno na jiné oblasti. Byl vytvořen soubor *TS – Compliance*, ve kterém byly definovány základní testovací scénáře zaměřené na vhodnost obsahu vzhledem k ratingu hry a na integraci Steam SDK.

První testovací scénář byl zaměřen na vhodnost obsahu a rating hry. Bylo zjištěno, že hra splňuje požadavky ratingových systémů pro všechny kategorie, takže by pro evropský trh mohla obdržet PEGI 3 a pro německý trh USK 0. V testovacím scénáři byla využita tabulka obsahující regionální ratingy (PEGI, ESRB, CERO, GRAC, CADPA, USK, ACIP) a byly provedeny kontroly propagačních materiálů. Nebyly nalezeny žádné závady. Omezení byla zaznamenána pouze v nemožnosti přístupu k dokumentaci a v nedostupnosti testování regionálních verzí (např. ruské či japonské).

Ve druhém TS bylo řešeno Steam SDK. Bylo zaměřeno na ověření správného fungování Steam funkcionalit, jako jsou achievementy, statistiky, kompatibilita se Steam Deckem, žebříčky a Cloud ukládání. Pro získávání těchto informací je využíváno Steam API, avšak pokud jsou osobní údaje (např. uživatelské jméno v žebříčcích) vývojáři předány, je podle GDPR považován za správce či zpracovatele těchto údajů. V EULA či zásadách ochrany osobních údajů je stanoveno, že musí být uživatelé informováni o tom, jaká data jsou zpracovávána, za jakým účelem a na jakém právním základě, a musí být zajištěna příslušná technická a organizační opatření k jejich ochraně. Ochrana dat je poskytována platformou Valve, avšak odpovědnost za osobní údaje, která jsou hře předána, nese vývojář.

Byla ověřena funkčnost všech testovaných prvků kromě kompatibility se Steam Deckem. Vzhledem k nemožnosti získat zařízení Steam Deck byly výsledky odvozeny z hlášení Steamu a testů provedených na ROG Ally Z1 Extreme. Informační hláška ze Steamu (Obrázek 54) upozorňuje na pouze částečnou kompatibilitu a doporučuje zvětšení textu pro lepší čitelnost na menších obrazovkách. Ačkoliv je Steam Deck vybaven 7,4" OLED displejem (LCD model 7"), při testování na 7" displeji ROG Ally Z1 Extreme nebyla nečitelná velikost textu zaznamenána. Bylo rovněž identifikováno, že po zadání textu je vyžadována softwarová klávesnice na obrazovce. V tomto případě by bylo vhodnější, aby byl text automaticky generován při ukládání a nešel měnit. Tím by byla zachována větší imerze pro přenosná zařízení. Přestože tyto nedostatky nejsou kritické, bylo doporučeno jejich odstranění při optimalizaci pro Steam Deck, aby byla zajištěna lepší propagace na platformě Steam a potenciálně zvýšeny příjmy a počet hráčů.



Obrázek 54 Steam Deck kompatibilita

(zdroj: Steam [70])

Bylo potvrzeno, že hra je vhodná pro všechny věkové kategorie a veškeré požadavky pro daný rating byly splněny. Integrace se Steam platformou proběhla úspěšně a hra byla ověřena jako hratelná i na přenosných zařízeních, s výjimkou několika nedostatků kompatibility se Steam Deckem. Doporučeno bylo, aby se na odstranění těchto nedostatků vývojáři zaměřili, čímž by byla hra platformou lépe vnímána a pravděpodobně i efektivněji propagována.

5.7 Automatizované testování

Automatizované testování nebylo považováno za primární směr, neboť se v studiu upřednostňovalo manuální testování, které dokáže odhalit i chyby unikající unit testům. Přesto byly pro zpracování konfiguračních souborů a ověření integrity konfiguračních dat navrženy automatizované testy. Ve složce *GameData* jsou uloženy konfigurační soubory pro zvířata, karty, habitaty a milníky s odměnami, a testování těchto souborů je důležité pro zajištění správného načítání a interpretace dat.

Testy byly rozděleny na 6 oblastí. Přesněji *Animals*, *BiomeManagerTests*, *Cards*, *Compendium*, *Habitats* a *MilestonesAndRewardsTests*. Každá skupina obsahuje dalších 1 až 5 testovacích případů, ve kterých jsou jednotlivé testy. Celkem lze těchto testovacích případů přes dvacet. Některé byly použity k výuce (např. *AnimalDrinkingStateTests*) a mohou být tedy i chybné nebo špatně optimalizované. V práci byly zmíněny jen některé z nich.

5.7.1 Unit testování

V rámci Unit testování byla testována třída *BiomesManager*, která jako persistentní singleton zajišťuje přístup k databázi biomů. Třída podle ID biomu vybere biom a vyše událost, aby další části hry s touto informací mohly pracovat. Obsahuje též statické metody pro překlad interních názvů biomů na oficiální názvy, viditelné ve hře a určení velikosti mapy.

```
public static string GetBiomeMapSize(int mapDifficulty)
{
    switch (mapDifficulty)
    {
        case 0:
            return "SMALL";
        case 1:
            return "MEDIUM";
        case 2:
            return "LARGE";
        default:
            Debug.LogError("Biome name not found");
            return null;
    }
}
```

Metoda *GetBiomeMapSize* byla definována statická funkce, v níž je na základě číselné hodnoty obtížnosti mapy *mapDifficulty* vrácen odpovídající řetězec popisující velikost mapy. Pro hodnoty 0 až 2 jsou vráceny přesně definované velikosti, zatímco pro jakoukoli jinou hodnotu je vráceno *null* a do konzole je zaznamenána chybová zpráva.

```
[Test]
public void GetBiomeMapSize_ReturnsCorrectValue()
{
    Assert.AreEqual("SMALL", BiomesManager.GetBiomeMapSize(0),
        "Mapa obtížnosti 0 by měla být SMALL");
    Assert.AreEqual("MEDIUM", BiomesManager.GetBiomeMapSize(1),
        "Mapa obtížnosti 1 by měla být MEDIUM");
    Assert.AreEqual("LARGE", BiomesManager.GetBiomeMapSize(2),
        "Mapa obtížnosti 2 by měla být LARGE");
}

[Test]
public void
GetBiomeMapSize_InvalidDifficulty_ReturnsNullAndLogError()
{
    LogAssert.Expect(LogType.Error, "Biome size not found");
    Assert.IsNull(BiomesManager.GetBiomeMapSize(3), "Metoda by
        měla vrátit null pro nevalidní mapDifficulty.");
}
```

Byl proveden test *GetBiomeMapSize_ReturnsCorrectValue*, jehož cílem bylo ověřit, zda statická metoda *GetBiomeMapSize* vrací očekávané textové popisy velikosti mapy pro úrovně obtížnosti 0 až 2. Prostřednictvím tří volání *Assert.AreEqual* byly porovnány návratové hodnoty pro vstupy 0, 1 a 2 s řetězci *SMALL*, *MEDIUM* a *LARGE*.

Druhý test *GetBiomeMapSize_InvalidDifficulty_ReturnsNullAndLogsError* byl zaměřen na chování při zadání neplatné hodnoty (v tomto případě 3). *LogAssert.Expect* nastavilo očekávání chybného zápisu do konzole a následně byla zavolána metoda *GetBiomeMapSize(3)* s neplatnou hodnotou. Metoda měla vrátit hodnotu *Null* a zapsat do konzole chybu. Testovací soubor dále obsahoval ten samý test pro metodu *GetBiomePublicName*, kde bylo ověřeno vložení správných a nesprávných jmen jednotlivých biomů.

```
[Test]
public void Deserialize_ShouldReturnNonNullDataAndCorrectCounts()
{
    var data =
        JsonUtility.FromJson<SerializedAnimalsImportData>(SampleJson)
        ;

    Assert.NotNull(data, "Deserializace vrátila null");
    Assert.IsNotNull(data.BiomeAnimalsData, "BiomeAnimalsData je null");
    Assert.AreEqual(1, data.BiomeAnimalsData.Count, "Neočekávaný počet biotopů");

    var biome = data.BiomeAnimalsData[0];
    Assert.AreEqual("TestBiome", biome.BiomeID);
    Assert.IsNotNull(biome.AnimalsData, "AnimalsData je null");
    Assert.AreEqual(2, biome.AnimalsData.Count, "Neočekávaný počet zvířat v biotopu");
}
```

Test *Deserialize_ShouldReturnNonNullDataAndCorrectCounts* byl navržen tak, aby ověřil deserializaci třídy *AnimalsImporter*. Při jeho provedení byl vzorový JSON řetězec (*SampleJson*) předán metodě *JsonUtility.FromJson<SerializedAnimalsImportData>* a bylo zjištěno, že výsledkem je nenulový objekt. Dále bylo doloženo, že obsahuje netriviální seznam záznamů pro biomy, přičemž byl nalezen přesně jeden záznam s ID *TestBiome* a v něm byly obsaženy dvě položky dat o zvířatech. Jakákoli odchylka (*null* návratová hodnota, chybějící pole *BiomeAnimalsData*, nesprávný počet biomů nebo zvířat) vede k chybě testu prostřednictvím příslušných volání *Assert*.

```
[Test]
public void Deserialize_AnimalEntries_HaveExpectedProperties()
{
    var data =
        JsonUtility.FromJson<SerializedAnimalsImportData>(SampleJson)
        ;
    var animals = data.BiomeAnimalsData[0].AnimalsData;

    var animalA = animals[0];
    Assert.AreEqual("AnimalA", animalA.ID);
    Assert.AreEqual("Forest,Field", animalA.PreferredHabitats);
    Assert.AreEqual("Marsh", animalA.AdditionalHabitats);
    Assert.AreEqual(2, animalA.Category);
    Assert.IsTrue(animalA.SpecialAnimal);

    var animalB = animals[1];
    Assert.AreEqual("AnimalB", animalB.ID);
    Assert.AreEqual("Desert,River", animalB.PreferredHabitats);
    Assert.AreEqual("", animalB.AdditionalHabitats);
    Assert.AreEqual(1, animalB.Category);
    Assert.IsFalse(animalB.SpecialAnimal);
}
```

Vzorový JSON je nejprve deserializován do objektu *SerializedAnimalsImportData*, poté je získán seznam *AnimalsData* a u každé položky jsou ověřena všechna klíčová pole – *ID*, *PreferredHabitats*, *AdditionalHabitats*, *Category* a *SpecialAnimal* – tak, aby bylo potvrzeno, že byla načtena a odpovídají očekávanému formátu i hodnotám.

5.7.2 Testování konfiguračních souborů

Byly provedeny EditMode testy čtyř konfiguračních souborů, přičemž bylo ověřeno načítání těchto souborů, zajištěna validace jejich obsahu a otestováno zpracování chyb v případě nesprávného formátu.

Jako příklad je vybráno testování souboru *AnimalsConfiguration.json*. Tento soubor je používán k načítání dat o zvířatech a jejich přiřazování k jednotlivým biotopům a je začleňován do importního procesu, ve kterém jsou data deserializována a následně využita pro vytvoření herních entit.

Pro zajištění správnosti načítání a validace dat byly vytvořeny testy pro třídy zajišťující tento proces, zejména pro třídu *AnimalsImporter*. Na základě konfiguračního souboru byly vytvořeny čtyři testovací soubory odpovídající jednotlivým biotopům, umístěné ve struktuře *Assets/Scripts/Tests/ImporterTests*, a byly pojmenovány podle příslušných biotopů (např. *AnimalsJurassicImporterTests*).

```
private const string ConfigPath =
"Assets/GameData/AnimalsConfiguration.json";
private SerializedAnimalsImportData data;
private static readonly object[][] AnimalTestCases = new[]
{
    New object[] {"Jurassic", "JurassicAmmonite", "JurassicReef",
0, false},
    // další objekty
};
```

Na začátku každého souboru je uvedena cesta k JSON souboru s konfigurací zvířat a je vytvořeno pole *data* typu *SerializedAnimalsImportData*, do kterého jsou po načtení deserializována všechna zvířata podle biomů. Následně je deklarováno statické read-only pole *AnimalTestCases*, kterým jsou parametrické testovací vstupy definovány. Každý jeho řádek obsahuje identifikátor biomu (např. *Jurassic*), jedinečné ID druhu, očekávané biotopy (řetězec biotopů oddělených čárkou), číselnou kategorii výskytu a příznak *isSpecialAnimal*.

```
[SetUp]
public void Setup()
{
    if (!File.Exists(ConfigPath))
        throw new FileNotFoundException($"Configuration file not
found at '{ConfigPath}'");

    string json = File.ReadAllText(ConfigPath);
    data = JsonUtility.FromJson<SerializedAnimalsImportData>(json);

    if (data == null)
        throw new InvalidDataException("Failed to deserialize JSON
into SerializedAnimalsImportData");
    if (data.BiomeAnimalsData == null)
        throw new InvalidDataException("BiomeAnimalsData is null in
the imported configuration");
}
```

Dále se v metodě označené *[SetUp]* nejdříve ověřilo, že na disku existuje konfigurační soubor a následně se celý obsah souboru vložil do řetězce. S pomocí *JsonUtility.FromJson<SerializedAnimalsImportData>* se data deserializovaly do objektu *data*. V případě chyby deserializace nebo chybějícího pole *BiomeAnimalsData* se vyhodilo *InvalidDataException*, pro zrušení dalších provedení testů.

```
[Test, TestCaseSource(nameof(AnimalTestCases))]
public void PreferredHabitats_NotEmpty(
    string biomeId,
    string animalId,
    string expectedHabitats,
    int expectedCategory,
    bool isSpecialAnimal)
{
```

```

var biome = data.BiomeAnimalsData.Find(b => b.BiomeID ==
biomeId);
Assert.IsNotNull(biome, $"Biome '{biomeId}' not found!");

var animal = biome.AnimalsData.Find(a => a.ID == animalId);
Assert.IsNotNull(animal, $"Animal '{animalId}' not found!");

Assert.IsFalse(string.IsNullOrEmpty(animal.PreferredHabi
tats),
    "PreferredHabitats should not be empty or whitespace!");
}

```

S využitím atributů `[Test, TestCaseSource(nameof(AnimalTestCases))]` je pro každý definovaný testovací případ (jednotliví dinosauři) ověřeno správné naplnění pole *PreferredHabitats*. Příslušný biome je vyhledáván podle *biomeId* a z něj je získána položka zvířete podle *animalId*. V případě nenalezení je vyvolána chyba NUnit. Následně je kontrolováno, že pole *PreferredHabitats* není *null*, prázdné ani sestavené pouze z mezer, čímž je zaručeno, že každému druhu je přiřazen alespoň jeden správný biotop.

```

[Test, TestCaseSource(nameof(AnimalTestCases))]
public void Category_InRange(
    string biomeId,
    string animalId,
    string expectedHabitats,
    int expectedCategory,
    bool isSpecialAnimal)
{
    var biome = data.BiomeAnimalsData.Find(b => b.BiomeID ==
biomeId);
    Assert.IsNotNull(biome, $"Biome '{biomeId}' not found!");

    var animal = biome.AnimalsData.Find(a => a.ID == animalId);
    Assert.IsNotNull(animal, $"Animal '{animalId}' not found!");

    Assert.Contains(animal.Category, new[] { 0, 1, 2 },
        "Category must be within the valid range [0,1,2]!");
}

```

Testem *Category_InRange* je zaručeno, že hodnota vlastnosti *Category* u každého druhu dinosauřích zvířat spadá do povoleného rozsahu. Nejprve je z konfiguračních dat vyhledán příslušný biome a v něm dané zvíře (v případě nenalezení je test označen jako selhání), poté je pomocí *Assert.Contains* ověřeno, že *animal.Category* nabývá pouze hodnot 0, 1 nebo 2. Tím je garantováno, že kategorie nikdy nepřekročí definovaný rozsah.

5.7.3 Spouštění testů

Po napsání testů je testerovi nebo vývojáři poskytováno několik možností jejich spuštění. První je manuální spuštění přes Test Runner (Obrázek 3), kde je prostřednictvím GUI (Graphical User Interface) určeno, zda budou spuštěny všechny testy, konkrétní sada nebo jednotlivé testy sekvenčně.

Druhá možnost spočívá ve spuštění testů skrze příkazový řádek. V něm musí být projekt uzavřen a nacházet se v adresáři s projektem. Vzhledem k tomu, že Unity nebylo přidáno do PATH, byla zadána úplná cesta k aplikaci spolu s parametrem *projectPath* (zastoupeným tečkou pro kořenový adresář projektu) a poté byl pomocí příkazu *runTests* zahájen Test Runner ve variantě *editmode*. Filtr byl nastaven na název příslušného souboru s testy, určilo se spuštění v *batchmode* bez UI a po skončení bylo nařízeno ukončení Unity. Výstup celého testu byl uložen do souboru *TestResults.xml*. Tento způsob lze automatizovat skrze CI/CD pipeline. V CI/CD scriptu (např. Azure Pipelines nebo GitLab CI) stačí tento krok vložit jako jeden job, zajistit přihlášení nebo instalaci správné verze Unity a poté zpracovat testovací reporty ve formátu XML či JSON podle potřeby.

Příkaz pro spuštění testů lokálně:

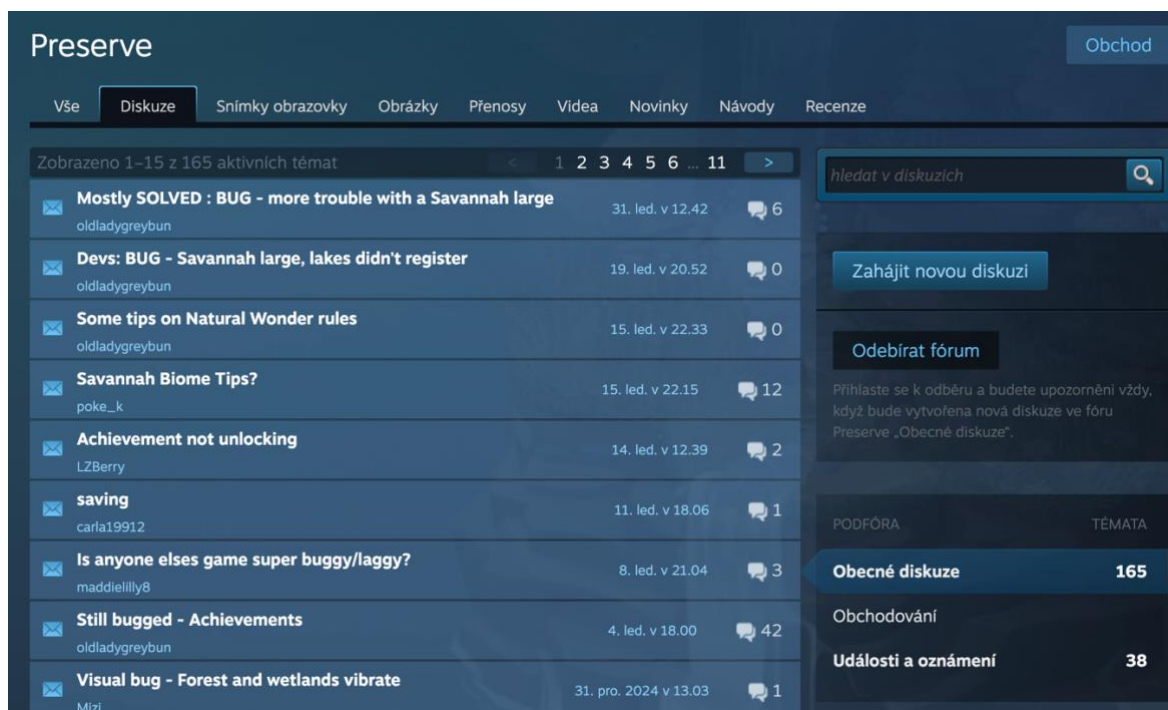
```
zdenek@Zdenek-MacBook-Air-2 preserve %  
"/Applications/Unity/Hub/Editor/2022.3.46f1/Unity.app/Contents/MacOS/  
/Unity" \  
-projectPath . \  
-runTests \  
-testPlatform editmode \  
-filter AnimalsEuropeanImporterTests \  
-testResults "/Users/zdenek/Unity/preserve/TestResults.xml" \  
-logFile testlog.txt \  
-batchmode \  
-quit
```

Bylo ověřeno Edit Mode unit testy, že zpracování konfiguračních souborů funguje bezchybně. Vzhledem k požadavku na detailní znalost herní logiky a programovacích dovedností nebyly automatizované testy primárně určeny běžným testerům, ale byly zaměřeny spíše na vývojáře a testery bílé skříňky. Jelikož byly vytvořeny až po dokončení příslušné části kódu a neočekávají se v blízké budoucnosti rozsáhlé změny, byly určeny především k průběžné kontrole kvality a k edukačním účelům.

5.8 Testování na základě zpětné vazby (Feedback testing)

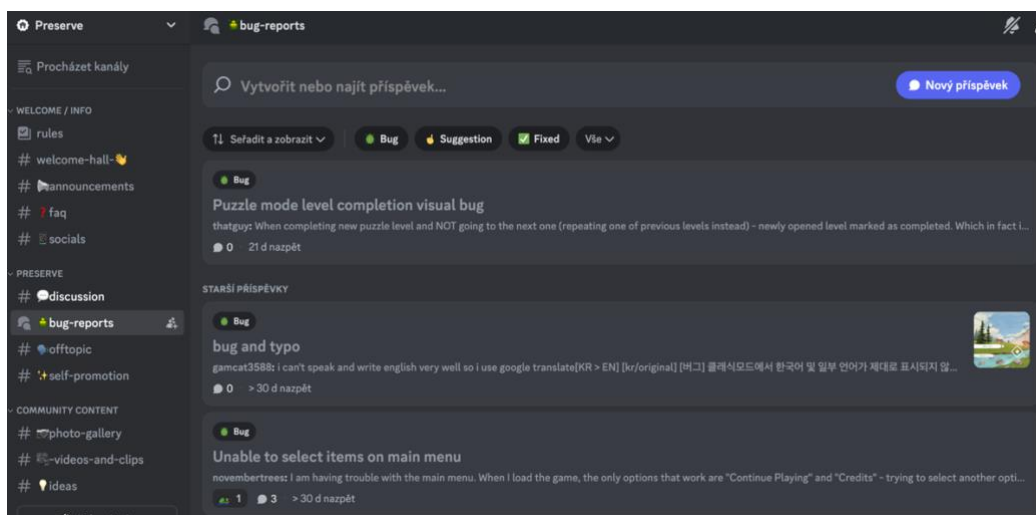
Hry jsou vytvářeny pro hráče a zpětná vazba od nich by měla být herními studií vyslechnuta, neboť může upozornit na chyby, které nebyly vývojáři ani testery odhaleny, a přispět tak k navržení úprav či vylepšení, jež budou hru lépe přizpůsobovat komunitě.

Zpětná vazba je sbírána formou diskusních skupin na platformách Steam a Discord (Obrázek 55 a Obrázek 56). Na Steamu nejsou diskuse filtrovány a slouží jak k hlášení chyb, tak k žádostem o pomoc či návrhům na vylepšení. Organizace diskusí je zde proto méně strukturovaná, avšak je umožněno psaní zpětné vazby bez nutnosti dalších aplikací. Diskuse jsou pravidelně procházeny vývojáři a na podněty je reagováno v případě potřeby. Nahlášené chyby jsou poté simulovány vývojářem, zaznamenány do ticketu v systému Jira, opraveny a otestovány testery. Oprava je v následující verzi nasazena a oznámena v příslušné diskusi.



Obrázek 55 Steam diskuse

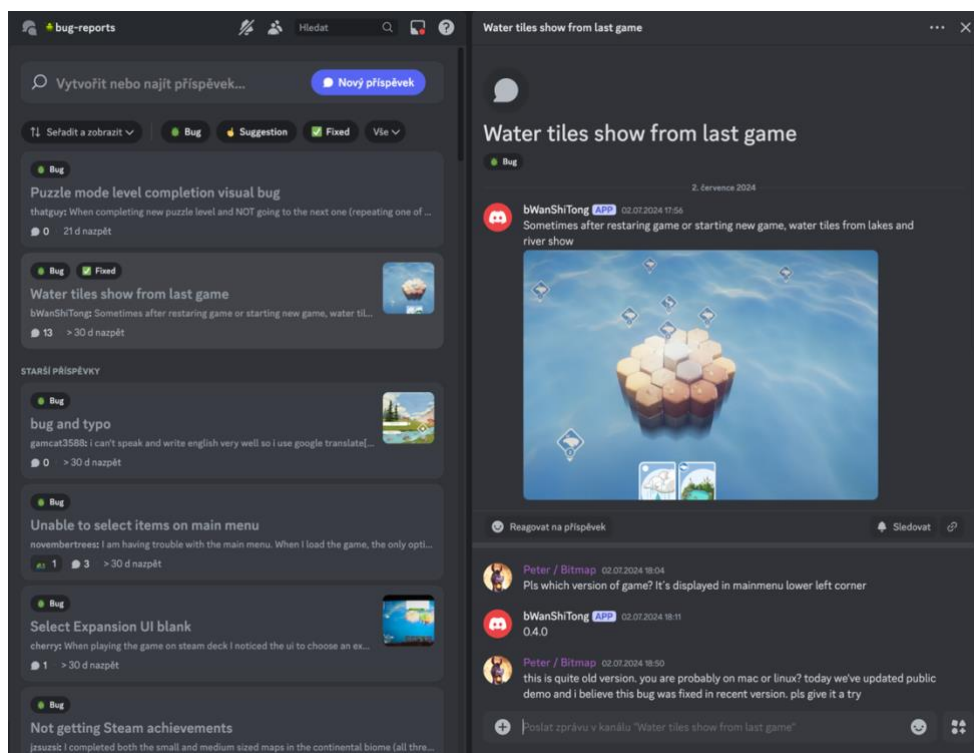
(zdroj: vlastní)



Obrázek 56 Discord kanál pro hlášení chyb

(zdroj: vlastní)

Na Discordu je hlášení prezentováno přehledněji. Kromě vyhledávání jsou přidávány tagy rozlišující chybu, návrh a stav opravy. Stejný princip jako na Steamu je zachovávan. Diskuse jsou vedeny převážně vývojáři, kterým jsou následně chyby přiřazovány k opravě nebo je poskytována asistence při řešení problémů. Obrázek 57 slouží jako příklad takové diskuse. Kdy hlášený problém byl opraven v novější verzi, kterou hráč neměl staženou.



Obrázek 57 Discord – chyba propisování vodních dlaždic z minulé hry

(zdroj: vlastní)

6 PRAXE V JINÝCH STUDIÍCH

V rámci rešerše byla provedena doplňková aktivita zaměřená na získání praktických poznatků z praxe jiných herních studií, jež byly následně využity při testování hry Preserve. Cílem bylo zmapovat postupy a přístupy k testování napříč různými týmy a prostředím a doplnit teoretickou část konkrétními zkušenostmi. Ke sběru dat byli osloveni zástupci studia NOXGames a SCS Software s.r.o., přičemž komunikace byla realizována prostřednictvím dotazníků a přímého kontaktu.

6.1 NOXGames

Brněnské indie studio bylo založeno v roce 2005 se zaměřením na mobilní hry. K aktuálnímu dni podle webu studio vydalo již přes 150 her. Byl vytvořen řetězec titulů zahrnující RPG, strategie, karetní i hypercasual žánr. Platformami pro jejich publikaci jsou Google Play a App Store. Mezi jejich tituly jsou uváděny Assassin Hero: Infinity Blade, Shadow Deck: Magic Heroes CCG, War Hex: 4X Válečná Strategie a Merge Neon Car: Idle Car Merge.

6.1.1 Dotazník

Jak organizujete testování ve Vašem herním studiu? Testování v naší firmě organizujeme primárně podle priorit releasů. Z důvodu rozsáhlého portfolia je nastavení priorit důležité. Mimo testování musíme počítat i s přímou asistencí game designu, UX/UI a developmentu samotného.

Jaké metody testování používáte při vývoji? Primárně využíváme metodu manuálního testování. Vzhledem k množství testovaných aplikací se díky této metodě dokážeme lépe přizpůsobit a během testování se zaměřit i na složitější aspekty mobilní aplikace. Metodu automatického testování jsme začali okrajově využívat na jedné z aplikací.

Jaký máte proces vývoje softwaru? A jak často podle něj testujete? Naše firma využívá spirálový vývoj. Celé QA oddělení se zapojuje do práce již od preprodukce, kde se soustředí primárně na průzkum konkurence. Následně celé oddělení vypomáhá s designerskou a produkční prací. Samotné testování začíná až u prvního prototypu.

Které typy testů používáte nejčastěji? Nejčastěji využíváme akceptační testy. U náročnějších projektů využíváme také integrační testy a systémové testy.

Jakým způsobem se vypořádáváte s regresním testováním při aktualizacích her?

Regresivní testování probíhá primárně u větších projektů, kde se s testováním začíná již při programátorské práci. Díky tomu dokážeme odhalit některé problémy již během vývoje a urychlíme tím proces vydání aktualizace.

Jak testujete multiplayer her? V našem portfoliu nemáme žádnou multiplayer hru.

Jaké máte postupy při testování na různých platformách (Android/iOS)? Po stupy při testování mobilních her máme pro platformy Android a iOS velice podobné. V obou případech se soustředíme na testování jak na testovacích zařízeních, tak i na reálných, kde se ujistíme, že aplikace funguje správně mimo testovací prostředí. V okrajových případech využíváme emulátory. U platformy Android se u určitých aplikací zaměřujeme také na testování aplikací na zařízeních s minimálním výkonem. Některé naše hry vydáváme i jako webové hry, v tomto případě je pro nás důležité testovat aplikaci ve větším spektru prohlížečů.

Testujete nějakým způsobem vytvořené módy z komunity? Komunita nevytváří módy.

Jaké výzvy jste měli v minulosti při testování? Naše největší výzva byla podle mě spojení testování našeho největšího projektu, strategické hry, s kompletním nastavením obtížnosti a herní ekonomiky.

Jak zajišťujete bezpečnost a stabilitu, zejména s ohledem na možné chyby nebo exploity? V rámci testování mobilních her se zaměřujeme na zajištění bezpečnosti a stability prostřednictvím pečlivého prověřování hry na různých zařízeních a operačních systémech. Kromě toho spolupracujeme s vývojáři na identifikaci a řešení nalezených problémů.

Jakou roli hraje automatizované testování ve vašem vývoji? Co za nástroje vy užíváte? Automatizované testování využíváme okrajově. Využíváme Unity Test Framework.

Jaké nástroje a technologie využíváte pro správu, sledování a dokumentaci testovacích procesů? Pro správu a sledování využíváme primárně ClickUp a Miro. Pro dokumentaci a tvorbu testovacích scénářů následně Google SpreadSheet.

Které oblasti testování jsou pro Vás prioritní (např. funkčnost, výkon, uživatelské rozhraní) a proč? Prioritní je pro nás především funkčnost samotné aplikace, která je podle mě nejdůležitější. Za další prioritní věc považujeme také výkon aplikace, protože vzhledem k cílovým skupinám některých aplikací musí být naše hry co nejlépe optimalizované.

Jakým způsobem testujete fyziku, správnost map, textur a objektů? U testování fyziky si určíme testovací scénáře, podle kterých následně postupujeme a odhalujeme chyby v chování. Správnost map, textur a objektů podle design dokumentů. Veškeré tyto aspekty testování konzultujeme s vývojáři.

6.1.2 Souhrn

Testování je ve studiu organizováno primárně podle plánovaných releasů, přičemž jsou využívány manuální i okrajově automatizované metody. Vývoj softwaru je prováděn podle spirálového modelu. Nejčastěji jsou aplikovány akceptační, integrační a systémové testy, přičemž regresní testování je zahajováno již během programátorské fáze. Ačkoli multiplayerové tituly v portfoliu chybí, byly testovány různé platformy (Android/iOS) velmi důkladně. Bezpečnost a stabilita jsou zajišťovány pečlivým testováním a úzkou spoluprací s vývojáři, přičemž automatizované testy jsou používány s Unity Test Framework pouze okrajově. Pro správu a dokumentaci testovacích procesů jsou používány nástroje ClickUp, Miro a Google Spreadsheet. Funkčnost a výkon aplikace byly deklarovány jako prioritní oblasti testování, zatímco fyzika, mapy, textury a objekty byly ověřovány prostřednictvím detailních testovacích scénářů a konzultací s vývojáři. Cílem studia je zajištění vysoké kvality produktů systematickým testováním a průběžnou spoluprací v rámci vývojového procesu.

6.2 SCS Software s.r.o.

SCS Software s.r.o. bylo založeno v roce 1997 v Praze a je primárně zaměřeno na tvorbu desktopových a konzolových her v žánru simulátorů. Mezi nejznámější tituly byly zařazeny Euro Truck Simulator 2, American Truck Simulator a World of Trucks.

6.2.1 Dotazník

Jak organizujete testování ve vašem herním studiu? QA team máme rozdělené na sub-teams, které se věnují dílčí části testování. Někteří testují přímo modely či konkrétní produkty, jiní se zaměřují na obecnou kontrolu kvality celistvého produktu. Celkově se poté řídíme dle blížícího se releasu a očekávané náročnosti testování.

Jaké metody testování používáte při vývoji? Máme však i automatické validační testy. Povětšinou přímé manuální testování.

Jaký máte proces vývoje softwaru? A jak často podle něj testujete? Kontrola kvality začíná již v preprodukcí. Během produkce následně dostáváme konkrétní tasky, které dedikovaný tester či skupina testerů otestuje. Čím blíže je daný projekt vydání, tím více testerů se na něj soustředí. Výjimkou však nejsou ani rychlé ad-hoc tasky.

Které typy testů používáte nejčastěji? Nejčastěji testujeme manuálně, neboť naším cílem je testovat hru tak, jak ji do rukou dostane zákazník-hráč.

Jak způsobem se vypořádáváte s regresním testováním při aktualizacích her? Vzhledem k tomu, že náš produkt je postaven na DLC a aktualizacích, má pro nás kompatibilita s již vydanými funkcemi vysokou prioritu.

Jak testujete multiplayer her? Abychom napodobili přirozené prostředí multiplayeru, scházíme se v určité dny ve skupinkách, ve kterých hrajeme multiplayer jako běžný hráč, s tím, že se soustředíme na potenciální problémy. Pokud je však plánovaná zcela nová funkce, je potřeba se kontrole kvality v multiplayeru věnovat stejně jako jakémukoliv jinému tasku.

Jaké máte postupy při testování na různých platformách (Windows/Linux/Mac)? Jak již byly zmíněny sub-teamy v QA, máme i specialisty na nejen jiné platformy (Mac, Linux), ale i na jiná zařízení (SteamDeck, VR).

Testujete nějakým způsobem vytvořené módy z komunity? Samotné módy netestujeme, ale občas je využijeme pro testování našich vlastních funkcí.

Jaké výzvy jste měli v minulosti při testování? Největší výzvou je najít balanc mezi kvalitou a časem, který má daný projekt k dispozici. Kvalita je pro nás samozřejmě prioritou, nicméně ve chvíli, kdy by se projekt odkládal i kvůli nejdrobnějším změnám, nemá taková práce žádný další přínos a nastavený systém je pak neefektivní.

Jak zajišťujete bezpečnost a stabilitu, zejména s ohledem na možné chyby nebo exploity? Bezpečnost kódu si kontrolují především sami programátoři. Stabilita a další se pak kontrolují jak manuálními, tak automatickými testy.

Jakou roli hraje automatizované testování ve vašem vývoji? Co za nástroje využíváte? Automatizované testování nevyužíváme jako primární možnost, nicméně je důležitou součástí naší kontroly kvality produktu. Veškeré nástroje, které využíváme jsou naše interní nástroje pro validaci a další testy.

Jaké nástroje a technologie využíváte pro správu, sledování a dokumentaci testovacích procesů? Využíváme Favro, Mantis a interní systémy.

Které oblasti testování jsou pro vás prioritní (např. funkčnost, výkon, uživatelské rozhraní) a proč? Nejvyšší prioritou je vždy stabilita. Jako další si klademe za cíl kvalitní zážitek pro samotné hráče, a jak oni nové funkce budou vnímat.

Jakým způsobem testujete fyziku, správnost map, textur a objektů? Jak již bylo zmíněno, testujeme primárně vše manuálně. Například u kontroly mapy využíváme testování po krátkých úsecích v iteračním systému, přičemž každé iteraci se věnuje jiný tester. Tím se zajistí nejen kontrola daného úseku, ale také to, že jej vidí různí lidé, z nichž každý si všímá jiných problémů. Podobně testujeme i ostatní produkty. Pomáhají nám také ony automatizované testy popsané výše.

6.2.2 Souhrn

Testování je ve studiu SCS Software s.r.o. organizováno prostřednictvím QA týmu rozděleného do menších týmů pro různé oblasti testování, přičemž jsou kombinovány manuální a automatizované metody. Kontrola kvality je zahajována již v preprodukcí a dále je udržována v průběhu celého vývoje, s důrazem na stabilitu produktu a kvalitu uživatelského zážitku. Pro správu testovacích procesů jsou využívány nástroje Favro a Mantis a jako nejvyšší prioritě je věnována stabilita produktu, následovaná kvalitou uživatelského prostředí. Manuální testování je aplikováno napříč herními aspekty – fyzikou, mapami, texturami a objekty – v rámci iterativního přístupu a s účastí různých testerů.

7 ZHODNOCENÍ TESTOVÁNÍ A DOPORUČENÍ

Od listopadu 2024 do března 2025 bylo na hře Preserve testováno s buildy od verze 0.6.70 do přibližně 0.8.19. Během tohoto období bylo identifikováno 75 chyb různých priorit – od drobných problémů, jako bylo zarovnání textu, až po chyby vysoké priority vyžadující optimalizaci textur či dalších procesů. Celkem bylo odehráno více než 160 hodin.

V této kapitole jsou prezentovány aplikované testovací techniky a průběh testování, přičemž je pozornost věnována cílům, použitým nástrojům a časové náročnosti. Následně jsou provedeny kvantitativní a kvalitativní analýzy výsledků a formulována doporučení pro další vývoj a testování.

7.1 Shrnutí průběhu testování a hodnocení testovacích technik

Testování hry Preserve probíhalo paralelně s jejím vývojem v rámci předběžného přístupu. Během tohoto procesu byly vydávány nové buildy, do kterých byly přidávány funkcionality, opravovány nalezené chyby a neúmyslně zaváděny chyby nové. Cílem bylo zajištění funkčnosti, stability, kompatibility a souladu s požadavky definovanými při návrhu. Kritické i okrajové chyby byly identifikovány před zpřístupněním hry ve verzi 1.0.

Při testování bylo využito několik různých testovacích metodik, které prověřovaly širokou škálu aspektů od ovládání, po grafiku až ke kvalitě hádanek. Jednalo se o následující techniky:

7.1.1 Funkcionální testování

V rámci funkcionálního testování se vytvořily čtyři excelové soubory s testovacími scénáři. Ty pokrývaly oblasti: chování základních prvků jednotlivých biomů, tutoriál, herní módy, herní bodování, herní prostředí, menu a nastavení (včetně UI), puzzle a ovládání. Testování bylo rozděleno na dvě základní úrovně.

Jedna z nejdéle trvajících a nejnáročnějších částí byla věnována vytvoření a otestování mechanik, přičemž do jejího dokončení bylo investováno přibližně 40–60 hodin. Vzhledem k absenci dokumentace bylo nezbytné věnovat desítky hodin samotnému pochopení všech herních mechanismů prostřednictvím průzkumu hry a konzultací s dalšími testery a vývojáři. Scénáře byly sestaveny ve formátu testovacích případů obsahujících předpoklady, postup a očekávaný výsledek, čímž byla umožněna jejich systematická aplikace a vyhodnocení.

Testování černé skříňky s prvky playtestingu bylo považováno za náročné zejména z hlediska interpretace očekávaného chování. Veškeré zjištěné nesrovnalosti byly protokolovány a sdíleny prostřednictvím Jiry či excelových souborů.

Jako hlavní negativa bylo identifikováno vysoké časové zatížení (v rámci desítek hodin), pozdní zapojení do vývojového procesu a absence podrobných dokumentací. Naopak byly vyzdviženy přínosy v podobě vytvoření komplexních testovacích scénářů, jež mohou sloužit nováčkům jako příručka pro rychlejší osvojení herních mechanik, a usnadněné zachycování a reprodukce chyb díky systematickému formátu testovacích případů.

7.1.2 Lokalizační testování

Pro účely lokalizačního testování byl vytvořen excel soubor se čtyřmi záložkami, které byly zaměřeny na klíčové oblasti ovlivňující kvalitu lokalizace: terminologie používaných výrazů a jejich konzistentní překlad napříč jazyky, vliv textových a grafických prvků na uživatelské rozhraní, formátování dat, čísel a času pro různé jazyky a obecné lokalizační aspekty, zahrnující kulturní správnost překladu, kontrolu diakritiky a kontext.

Testování lokalizace bylo provedeno během přibližně 10–12 hodin, v nichž byly vytvořeny testovací scénáře, provedeny rešerše specifik jednotlivých jazyků a realizováno samotné testování. Vzhledem k zaměření na českou a anglickou mutaci byla časová náročnost omezena. Zahrnutí dalších jazyků by v případě absence plynulé znalosti testerem výrazně prodloužilo dobu potřebnou pro ověření.

Pro překlady a jazykovou korekturu byly nasazeny nástroje DeepL a Grammarly, přičemž jejich použitím byla zajištěna rychlá konverze mezi češtinou a angličtinou a automatická kontrola pravopisu a gramatiky. V některých případech byl pro ověření významu a jazykového stylu využit i nástroj ChatGPT.

Po vytvoření testovacích scénářů byla největší časová náročnost vynaložena na ověření všech částí hry, ve kterých bylo prověřeno, zda jsou textace správně přeloženy, konzistentní a zda je uživatelské rozhraní graficky přizpůsobeno. Obzvláště náročné bylo testování Kompendia, jež bylo nutné projít v obou jazykových mutacích a porovnat jeho významové i vizuální zpracování. Mezi nejčastěji zaznamenané závady byly zařazeny nesprávně zarovnané texty, nekonzistentní překlady používaných termínů a překlepy.

Testování bylo prováděno i na různých zobrazovacích zařízeních (7", 13", 15" a 27" displeje při různých rozlišeních), čímž byla zvýšena časová náročnost. Bylo zjištěno, že lokalizace často ovlivňuje délku textů, a tím i rozvržení prvků uživatelského rozhraní.

Mezi negativní dopady bylo zařazeno vysoké časové zatížení testu a požadavek na pokročilou znalost cílových jazyků. V menších týmech bylo proto zvažováno využití strojového překladu s následným zapojením komunity k nahlášení a opravě chyb nebo postupné přidávání jazyků až po dokončení kompletní lokalizační optimalizace.

Naopak bylo zjištěno, že testování je intuitivní a může být prováděno prakticky kterýmkoli členem týmu s rodilou nebo vynikající znalostí daného jazyka. V rámci české a anglické lokalizace byly většinou odhaleny drobné chyby, které mohly ovlivnit uživatelský zážitek.

7.1.3 Testování kompatibility

Bylo ověřeno chování hry na různých hardwarových a softwarových konfiguracích v rámci minimálních i doporučených systémových požadavků. Testování bylo provedeno na čtyřech zařízeních s rozdílným výkonem a operačními systémy – MacBook Pro (2015), MacBook Air M1 (2020), Acer Aspire E1-531 a HP ProBook 450 G5. Detailní specifikace byly uvedeny v kapitole Testování kompatibility (Compatibility testing). Některé další aspekty, jako různá rozlišení obrazovky či odlišné poměry stran, byly zařazeny do výkonnostního nebo funkcionálního testování.

Testování kompatibility bylo stráveno přibližně 8–10 hodin. Nejvíce času bylo věnováno zajištění vhodného hardwaru a jeho přípravě (instalace operačního systému, Steamu a testované aplikace). K monitorování výkonnosti hardwaru byly využity nativní nástroje operačních systémů – Správce úloh a Monitor aktivity. Pro podrobné sledování byla nasazena utilita Pidstat a snímková frekvence včetně využití paměti byla zaznamenávána interním vývojářským nástrojem.

Dlouhá příprava pro testování byla způsobena tím, že doporučené hardwarové konfigurace podle platforem (např. Steam průzkum HW a SW) nebyly k dispozici. U malých vývojářských týmů jsou hry často optimalizovány pouze pro vlastněná zařízení a aktualizace jsou v čase postupně implementovány. I přesto byly testováním kompatibility získány důležité poznatky.

Po přípravě zařízení bylo testování provedeno bez výrazné časové náročnosti. Naměřené výsledky byly snadno měřitelné a porovnatelné. Díky využití odlišného hardwaru byly odhaleny optimalizační závady, které by na výkonnějších konfiguracích zůstaly skryty, čímž byl přispěn k vylepšení výkonu a stability hry napříč platformami. Pro základní monitorování aktivity hry byly nasazeny nativní nástroje (Správce úloh, Monitor aktivity), což bylo shledáno dostačujícím, zatímco tvorba reportů a záznamů byla pracnější. Automatický monitoring byl realizován pomocí nástroje Pidstat, jenž zapisoval data do logů převoditelných do CSV či TXT formátu. Záznamy byly tímto zjednodušeny, avšak jejich následné zpracování vyžadovalo dodatečnou automatizaci skriptů či externími nástroji.

Mezi negativní aspekty bylo zařazeno požadování široké škály hardwarového vybavení pro testování, což může pro menší týmy představovat problém. Dále byla identifikována vysoká časová náročnost na přípravu testovacího prostředí a potřeba opětovného provádění testů po každé výraznější změně výkonu či optimalizace. V důsledku toho se proces stával zdoluhavým a byl náchylný k časté replikaci testování i jinými metodami, například regresními nebo výkonovými.

7.1.4 Výkonnostní testování

V rámci výkonnostního testování byla věnována pozornost spotřebě baterie a vykreslovacímu výkonu, testu výdrže i reakci na vnější přerušení. Pro měření byly nasazeny nástroje Správce úloh, Monitor aktivity a interní nástroj pro sledování FPS a využití paměti. Testována byla zařízení ROG Ally Z1 Extreme, HP ProBook 450 G5 a MacBook Air M1. Celková doba testování dosáhla přibližně 30 hodin. Nejvíce času bylo vyčleněno testům spotřeby a vytrvalosti (každý trval kolem 12 hodin) a zbývající 4 hodiny byly věnovány testu přerušení.

Při testování spotřeby baterie a vykreslovacího výkonu byla hra provozována na handheld konzoli, u níž byl využit nástroj Command Center pro úpravu spotřeby a výkonu. V rámci tohoto nástroje bylo možné konfigurovat spotřebu, rozlišení, zobrazování FPS a aktivovat režimy pro dočasné zvýšení výkonu či kvalitnější vykreslování. Testování bylo provedeno v režimu výkonu i režimu výdrže až do úplného vybití baterie. Časová náročnost byla zapříčiněna opakovaným trojím spuštěním a omezenou dostupností testovacího zařízení. Během testu byly odhaleny problémy s optimalizací při práci na rozsáhlejších mapách, které způsobily nadměrné zatížení systému. Tyto závady byly vyhodnoceny jako závažné a přiřazena jim byla vysoká priorita.

Chod hry při dlouhodobém spuštění byl ověřován testováním vytrvalosti. Hra byla otevřena více než jedenáct hodin a stavy nečinnosti a aktivity se střídaly. Během testu žádné chyby nezaznamenány nebyly, avšak zpětně lze konstatovat, že by bylo efektivnější, kdyby byl tento čas využit i k souběžnému provádění dalších testů (např. regresních).

Chování hry při externích zásazích (například odpojení ovladače nebo uspaním zařízení) bylo provedeno testem přerušení. Testování bylo oproti předchozím dvěma zdaleka rychlejší a jediný problém byl zaznamenán při přechodu z režimu celé obrazovky do okna, kdy došlo k výraznému snížení FPS.

Měřitelnost výsledků, jejich přesnost a dobrá opakovatelnost jsou obecnými výhodami výkonnostního testování. Detailní data jsou v případě spotřeby a FPS získávána díky vhodným nástrojům a následně používána k optimalizaci hry.

Za hlavní nevýhodu je považována časová náročnost. Test vytrvalosti a spotřeby nelze výrazně optimalizovat, a většina práce je proto prováděna manuálně. Dostupnost specifického hardwaru může být pro menší týmy opět problémem a u testu přerušení je z důvodu nepředvídatelných vnějších vlivů hrozba vyššího rizika falešně pozitivních či falešně negativních výsledků.

7.1.5 Testování dodržování předpisů

Pro úspěšné vydání hry na cílových platformách a v jednotlivých regionech je testování dodržování předpisů považováno za nezbytné. Ověření souladu s pravidly dané platformy (např. Steam), s věkovými ratingy, s legislativními požadavky i s technickými specifikacemi, jako je integrace služeb dané platformy (např. Steam SDK), je součástí tohoto procesu.

Plnohodnotné testování přímo přes Steamworks rozhraní nemohlo být provedeno, jelikož přístup k němu nebyl k dispozici. Existence hry na platformě Steam po delší dobu však naznačuje, že základní podmínky vystavení musely být již dříve splněny. Testování bylo proto vykonáváno pouze formou omezené validace a rešerše, s cílem demonstrovat, jak by měl správný compliance proces vypadat.

Přibližně šest hodin bylo věnováno samotnému testování. Nejvíce času bylo vyžádáno rešerší k jednotlivým ratingovým skupinám (např. ESRB, PEGI), dokumentací Steam SDK a souvisejícími právními aspekty. Vytvoření testovacích scénářů a jejich následná kontrola byly již časově méně náročné, avšak ve výsledku bylo prokázáno, že testování nebylo klíčové a že investovaný čas mohl být využit efektivněji jinde.

V rámci dodržování předpisů byly výsledky většiny testů vyhodnoceny pozitivně.

Obsah hry byl vyhodnocen, že odpovídá minimálním požadavkům pro nejnižší věkové hodnocení (neobsahuje vulgarismy, explicitní násilí ani kulturně citlivý nebo nevhodný obsah). Bezproblémové využívání základních funkcí, jako jsou achievementy, žebříčky a cloudové ukládání, bylo testováním Steam SDK potvrzeno. Problém byl zaznamenán pouze u kompatibility se zařízením Steam Deck, konkrétně se zobrazováním softwarové klávesnice a velikostí textu. Při testování na podobném zařízení (ROG Ally) však nebyl žádný problém zjištěn.

Nevýhody této testovací techniky jsou především spojeny s náročností na pochopení legislativních norem. Jako další komplikace je považována nedostupnost přímého přístupu k administrátorským rozhraním (např. Steamworks), která jsou nutná pro plnohodnotné testování. Problémem může být také vysoká náchylnost k chybám způsobená nesprávným pochopením pravidel a norem. V důsledku tohoto rizika je často navazována spolupráce herního studia s externími partnery, kteří tento proces profesionálně zastřešují.

Hlavní výhodou testování dodržování předpisů je robustní dokumentace. V tomto případě byla použita dokumentace platformy Steam, jež poskytuje přehledné informace k technickým i obsahovým požadavkům. Další výhodou je ověření připravenosti hry pro širší distribuci a minimalizované riziko zamítnutí publikace ze strany platformy.

7.1.6 Unit testování

V rámci Unit testování byly primárně napsány EditMode testy pro kontrolu zpracování konfiguračních souborů či nastavení biomů a obtížnosti. Testování se týkalo konfiguračních souborů pro zvířata, karty, habitaty a milníky s odměnami. Jelikož byly testy implementovány až po vytvoření tříd pro zpracování souborů, byly jejich účelem regresní testování a edukační účely.

Samotné psaní testů bylo dokončeno za zhruba 20 hodin. Projekt však původně na unit testování připraven nebyl, protože ve studiu se testy často nevyužívaly. Byly proto provedeny dodatečné úpravy, které zahrnovaly například konfiguraci *asmdef* souborů. Tato fáze, včetně řešerše ke vzniklým problémům, trvala přibližně dalších 4 až 6 hodin.

Rychlost provedení je ale někdy kritickým faktorem důvodu jejich zavedení. Manuální kontrola konfiguračního souboru by totiž znamenala spuštění hry, následné zapnutí příslušného módu a ověření správnosti veškerých dat, což může v závislosti na rozsahu

kontroly zabrat pět a více minut. Unit testy sice mohou vyžadovat jednu až dvě hodiny pro sepsání a ověření, ale poté jsou spouštěny velmi rychle a výsledky jsou známy během několika sekund.

Pro psaní testů byl využit nástroj Visual Studio Code a testy byly spouštěny skrze Test Runner v prostředí Unity. Pro systematictější nasazení unit testování by bylo vhodné použít i nástroj Code Coverage Package. Díky němu je možné sledovat historii pokrytí kódu, zobrazit seznam metod s jejich procentuálním pokrytím a při každém spuštění testu nebo buildu aplikace generovat HTML reporty. Tento nástroj však nebyl využit, jelikož by přesahoval cíle a rozsah práce.

Jako užitečný doplněk se ukázalo využití umělé inteligence. Pomocí nástrojů jako ChatGPT a Microsoft Copilot mohou být provedeny optimalizace kódu, navrženy jednoduché testy a začínajícímu vývojáři či testerovi může být vysvětleno chování metod či celých tříd.

Měřitelnost výsledků, čitelnost testů, rychlost exekuce a možnost snadné automatizace skrze CI/CD nástroje jsou považovány za silné stránky unit testování. Do budoucna by bylo vhodné, aby byly testy napojeny například přes Unity Cloud Build.

Testeři černé skříňky nejsou přizpůsobeni k těmto testům a je vyžadováno alespoň základní technické vzdělání. V důsledku časové náročnosti implementace i údržby a s ní spojených nákladů nebyly unit testy dosud zavedeny a byly zaváděny později.

Ačkoliv není unit testování považováno za nezbytnou součást testovacího procesu, může být na vhodných místech výrazně přispěno ke zvýšení stability a udržitelnosti.

7.1.7 Testování na základě zpětné vazby

Při dostatečně aktivní a početné herní komunitě je možné získat velké množství externích testerů. Zpětná vazba je poskytována prostřednictvím cíleného playtestingu nebo volného ad hoc testování. Chyby jsou zpravidla odhalovány buď ty nejčastější, nebo pouze ty okrajové, které by běžně nebyly testovány.

Je ověřováno, zda hlášené chování skutečně nežádoucí je. V rámci těchto kontrol jsou simulovány hlášené situace a v případě potvrzení problému je do Jiry vytvořen ticket. Po opravě je hráčům zaslán komentář o tom, že byla chyba opravena v konkrétní verzi. Duplicitní hlášení jsou nejčastěji odhalována přímo v komunikačních kanálech, ve zbývajících případech je duplicita zjišťována až po založení chyby v Jiře.

Chyby byly zaznamenávány buď na komunitním Discord serveru, nebo na Steam stránce v sekci Diskuse. Za hlavní výhodu Steamu byla považována jeho dostupnost, neboť pro nahlášení chyby nebyla vyžadována žádná další aplikace – stačilo vytvoření nové diskuze s popisem problému. Možnost lepší filtrace výsledků a správy třídění nahlášených chyb však chyběla. Tento nedostatek byl vyřešen Discord serverem, v němž jsou k dispozici pokročilé nástroje pro organizaci hlášení (řazení do tagů podle stavu či typu komentáře) a zároveň je poskytován přímý kanál pro komunikaci mezi týmem a hráči. Vzhledem k tomu, že Discord není dostupný všem hráčům, je nutné oba kanály kombinovat.

Za typické problémy při komunitním testování byly považovány nedostatečné popisky chyb, nutnost zpětného kontaktování hráčů pro doplňující informace a situace, kdy byla používána stará verze hry. Mezi hlavní nevýhody je řazena nutnost vybudování dostatečně početné komunity hlásící chyby a zpravidla správa více komunikačních kanálů pro budování komunity a získávání nových hlášení.

Naopak za výhody byly považovány nízká finanční náročnost, široký záběr testovacích scénářů, vyšší počet „externích testerů“, včasné zachycení problémů v právě vydané verzi a získávání návrhů na vylepšení a optimalizaci přímo od koncových hráčů. Velmi pozitivní hodnocení hry bylo od hráčů získáváno, které bylo zapříčiněno rychlou komunikací a opravami nahlášených chyb.

7.2 Analýza výsledků testování

Jako zdroj všech výsledků a analýzy byl použit excelový soubor *Bug report*, v němž je uveden seznam všech nalezených chyb, jejich stav k verzi 0.9.16 a veškeré analytické grafy a tabulky.

7.2.1 Kvantitativní výsledky

Byla provedena kvantitativní analýza na celkovém počtu 73 nalezených chyb, které byly nalezeny během testovacího období. Pro účely vyhodnocení byly chyby klasifikovány podle priority (závažnosti), kategorie výskytu a kombinace obojího. Data byla následně zpracována pomocí kontingenčních tabulek a základní deskriptivní statistiky v aplikaci Microsoft Excel.

Chyby byly rozděleny podle priorit do tří kategorií: *Low*, *Medium* a *High*. Z Tabulka 7 vyplývá, že skoro 63 % je nízké priority a nebrání hráčům nijak ve hraní. Do kategorie vysoké priority bylo zařazeno 15 % chyb, jejichž výskyt může být vnímán jako negativní

vliv na herní zážitek. Obecně byly naměřené hodnoty hodnoceny jako odpovídající stavu hry v předběžném přístupu a podle recenzí nebyl zjištěn žádný negativní dopad na hru.

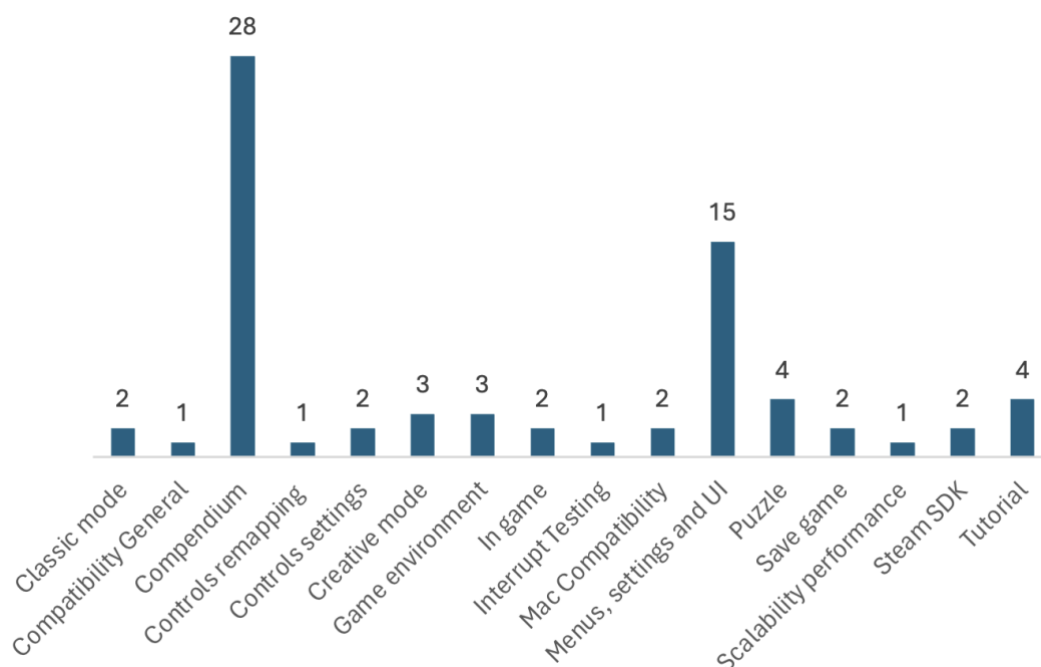
Tabulka 7 Tabulka rozložení dle priorit

Low	Medium	High
46	16	11
(63,01 %)	(21,91 %)	(15,07 %)

(zdroj: vlastní)

Počet chyb rozříděných podle kategorie je charakterizován grafem níže (Obrázek 58). Z celkových 73 bylo nejvíce chyb zaznamenáno v Kompendiu (28 chyb – 38,36 %). Druhá nejvyšší poruchovost byla zjištěna v sekci Menu, nastavení a UI (20 chyb, 55 %) a na třetím místě byly evidovány hádanky (5,48 %). Priority, jež jsou převážně nízké nebo střední, v grafu reflektovány nejsou.

V případě Kompendia byly odhaleny chyby v zarovnání textu a chybné výrazy. V druhém nejvyšším případě bylo zjištěno jak nesprávné rozložení UI elementů, tak prodleva v zobrazení textu či nevhodného nastavení kvality textur.



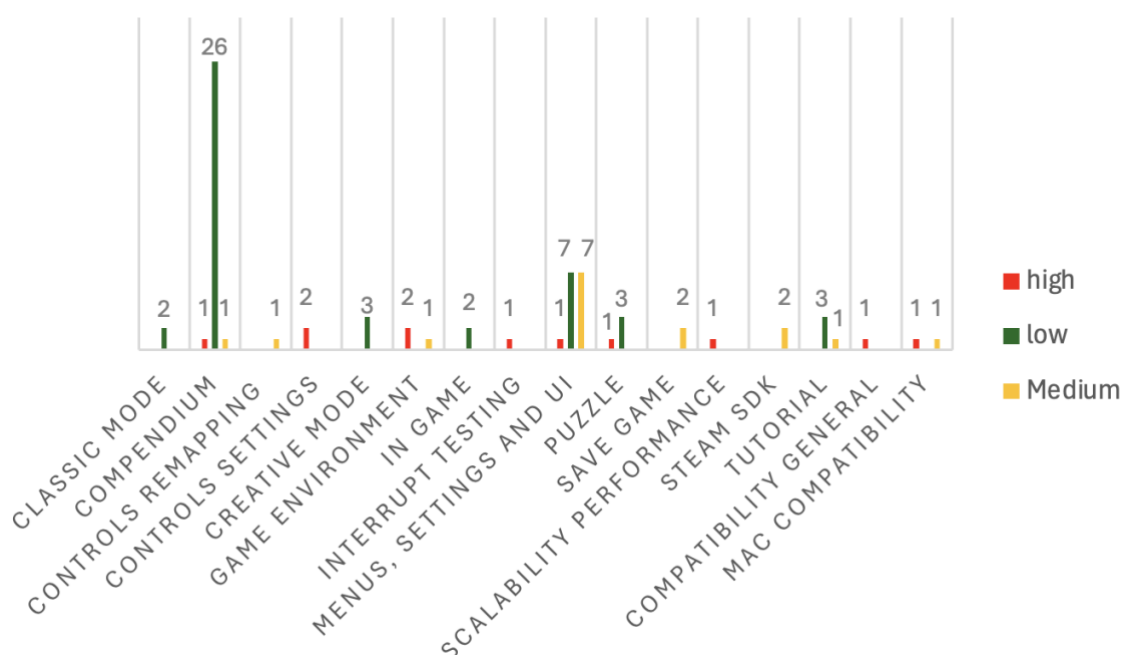
Obrázek 58 Graf počtu chyb podle kategorií

(zdroj: vlastní)

Byl vypočten podíl chyb s nejvyšší prioritou pro každou kategorii. Nejvíce závažných chyb bylo soustředěno v kategoriích *Controls settings*, *Interrupt Testing*, *Scalability performance* a *Compatibility General*, kde činil podíl 100 %. V těchto případech se jednalo výhradně o chyby s vysokou prioritou, bez výskytu nižších priorit.

Na druhém místě byl zaznamenán podíl 66,67 % v kategorii *Game environment*, kde byly dva závažné bugy doplněny jedním bugem střední priority. Třetí místo zaujala kategorie *Mac compatibility* s 50 % podílem, tedy s jednou chybou vysoké priority a jednou střední priority. Podíl 25 % byl vypočten pro *Puzzle*, kde vedle vysoce prioritní chyby byly zaznamenány tři chyby nízké priority. V sekci *Menus, settings and UI* činil podíl 6,67 % (jedna vysoce prioritní chyba mezi sedmi nízko a sedmi medium prioritami). V kategorii *Compendium* byl zjištěn podíl 3,57 %, kdy jedna chyba vyšší priority byla přítomna mezi více nízko-prioritními chybami. V ostatních oblastech (*Classic mode*, *Controls remapping*, *Creative mode*, *In game*, *Save game*, *Steam SDK* a *Tutorial*) žádná vysoce prioritní chyba zaznamenaná nebyla. Na základě těchto dat by měly být prioritně řešeny sekce s vysokým podílem závažných chyb a zároveň s ohledem na náročnost jejich odstranění.

Z tabulky počtu chyb podle kategorie a priorit (Obrázek 59) je odvozen graf. Pokud jsou složitost a časové vytížení opomenuty, může být týmu efektivně předložen tento graf pro organizaci oprav a následné testování daných kategorií.

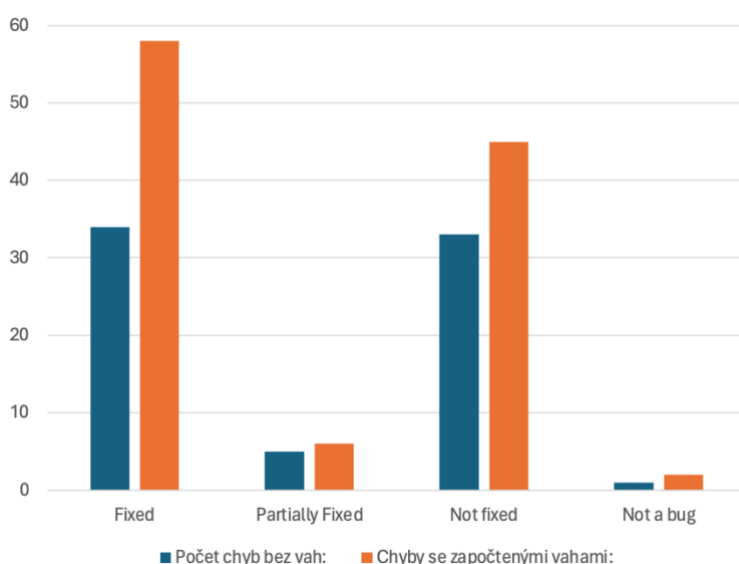


Obrázek 59 Graf počtu bugů podle kategorie a priority

(zdroj: vlastní)

Při ověření poslední verze ze začátku května (build 0.9.16) bylo testováním odhaleno, že ze 73 nalezených chyb bylo 34 opraveno. 33 chyb zůstalo neopraveno a 5 chyb bylo opraveno pouze částečně. V jednom případě nebyla identifikována chyba, ale špatná interpretace výsledků (BUG059).

Výsledek je zobrazen Obrázek 60 a jeho modrým grafem. Oranžovým grafem je znázorněno započítání váhy priorit do výsledků (1 = low, 2 = medium, 3 = high). Po započtení priorit byl zjištěn rozdíl mezi opravnými a neopravnými chybami ve výši 13 bodů, což dokládá, že při opravách byly upřednostněny chyby s vyšší prioritou a nízké priority byly ponechány stranou. Ze 33 neopravených chyb jich bylo 20 s nízkou prioritou. Jako další faktor byly uváděny přípravy na vydání verze 1.0 a portování na konzole.



Obrázek 60 Graf stavu chyb ve verzi 0.9.16

(zdroj: vlastní)

Na obou platformách byl zaznamenán pokles počtu hlášených chyb. Na platformě Discord bylo za posledních 30 dní hlášeno 2 chyby a za posledních 60 dní jich bylo zaznamenáno 5. Na Steamu bylo vytvořeno 3 až 6 nových diskusí měsíčně, přičemž jako nahlášené chyby bylo označeno pouze 1 až 2 příspěvky měsíčně.

Graf počtu hráčů hry Preserve za poslední 3 měsíce je znázorněn na Obrázek 61. Z grafu je patrné, že se počet hráčů za toto období neměnil a pohyboval se průměrně mezi 11 až 14. Hře bylo na Steamu uděleno 349 pozitivních hodnocení a 23 negativních. Poslední negativní hodnocení byla vytvořena v dubnu a únoru a byla uplatněna výhradně kvůli odstranění ruské jazykové mutace.



Obrázek 61 Steam statistiky hry Preserve

(zdroj: Steam charts [74])

7.2.2 Kvalitativní výsledky

Na kvantitativní výsledky je v následující kapitole navázáno a bude poskytnuto hlubší porozumění nalezeným chybám a jejich vlivu na uživatelský zážitek. Z kvantitativního zpracování lze identifikovat několik oblastí, v nichž byly chyby zjištěny, a na jejich souvislosti a podrobnější charakteristiku se bude zaměřeno.

Lokalizační a jazykové nedostatky byly vymezeny jako chybějící nebo nesprávné překlady, gramatické omyly a nesourodá terminologie napříč hrou. Chyby samy o sobě nejsou zpravidla považovány za kritické, avšak je jimi ovlivňována profesionalita a je tímto snižována lokalizační kvalita. Jelikož byl do hry zajištěn překlad do jedenácti jazyků, jsou tyto chyby považovány za nezbytný prvek v průběhu vývoje.

Do této kategorie bylo zařazeno nejvíce nahlášených chyb ve dvou skupinách. V první skupině se jednalo o *chybějící nebo nesprávné překlady*, kterých bylo zaznamenáno 17 případů výskytu (např. BUG001 nebo BUG013). Je třeba zdůraznit, že kontrola se týkala české lokalizace a vzhledem k tomu, že studio sídlí na Slovensku, byly v některých případech v české verzi ponechány slovenské texty či naopak anglické výrazy nepřeložené.

Druhá skupina, rovněž s 17 výskyty, zahrnovala *gramatické chyby* (např. BUG006 nebo BUG012). Ačkoliv se nejedná o závažné chyby, může být kvalita lokalizace na daném trhu vnímána jako nízká a dochází k narušení čtivosti popisků. Častými chybami byly nedostatky v interpunkci, nesprávné skloňování a opomenutí teček či čárek.

Méně často byly zaznamenány chyby související s *nesourodou terminologií* (např. BUG008 nebo BUG014). Pojmy jako div (ang. wonder) nebo biotop (ang. habitats) byly v hře používány, avšak byly zjištěny i tvary jako div divů nebo habitaty, které mohou hráče mást. Jako další příklad bylo překladem označení šestiúhelníkové dlaždice jako dílek. Stejně jako u gramatických chyb se tímto prokázala neprofesionalita lokalizace a možnost zmatení při čtení popisků. Tyto chyby jsou často způsobovány strojovým překladem.

UX/UI nedostatky jsou řazeny mezi méně závažné chyby, mohou však hru vyobrazovat neprofesionálně a ovlivňovat zážitek hráče. Typicky jsou do těchto chyb zahrnovány různá nezarovnání textu, rozmazané, neostré či chybějící grafické prvky a neintuitivní ovládání.

Čtrnáctkrát bylo zaznamenáno *nezarovnání textu* (např. BUG002 nebo BUG007) kdy všechny případy kromě dvou byly opraveny. Nejednotné odsazení textu, nevhodné zalomení řádků a vizuálně posunuté prvky byly identifikovány jako příčiny, jimiž byla narušena konzistence při hraní. Nejčastěji byl tento problém pozorován v Kompendiu nebo v nastavení hry.

Rozmazané, neostré či chybějící grafické prvky byly zjištěny v pěti případech (např. BUG011 a BUG047) a jednalo se o chybějící ikony nebo neostré prvky (habitat UI, stromy, tráva a zvířata) při nízké kvalitě či rozlišení.

Neintuitivní ovládání bylo zaznamenáno ve třech případech (např. BUG064 a BUG065), kdy byla očekávána opačná reakce při invertování kláves pro přiblížení nebo náklon, než byla defaultně nastavena.

Přímý dopad na hraní hry nebo dosažení herních cílů je způsobován *funkčními chybami*. Přímé selhání mechanik hry, které je vyvoláno těmito chybami, vede k nesprávnému nebo nefunkčnímu chování. Mezi uvedené problémy patří nesprávné fungování žebříčků v určitých situacích, ovládání hrací plochy během aktivního menu, nefunkční hudba v pozadí při hraní či možnost zadání speciálních znaků při ukládání hry a jejich následné chybné zobrazení.

Celkem bylo evidováno 16 případů (např. BUG055 a BUG056) a u některých z nich bylo zjištěno překrytí do jiných kategorií, což dokládá tomu, že jedna chyba může mít více dopadů najednou.

V sedmi případech byly zaznamenány **problémy kompatibility a optimalizace** (například BUG070 či BUG071), které zahrnovaly neúspěšné spuštění nebo předčasné

ukončení hry na určitých hardwarových sestavách a snížení FPS při specifických grafických nastaveních či rozsáhlém herním plánu. Jedná se o závažné chyby, jejichž odstranění je nezbytné pro zpřístupnění hry širšímu okruhu hráčů a pro zachování pozitivního herního zážitku.

Testování výrazně přispělo k nalezení a eliminaci kritických chyb ve hře. Z Obrázek 60 a ze souboru *Bug report* vyplývá, že kromě dvou kritických chyb byly všechny zbývající chyby opraveny a že regresním testováním neobjevily nové problémy.

Díky testování byly rovněž odhaleny četné středně a nízko prioritní chyby, jež ovlivňují uživatelský dojem ze hry. Bylo rozhodnuto, že se nejprve upřednostní dokončení anglické verze, příprava verze 1.0 a tvorba portů na konzole. Opravy odhalených chyb budou prováděny až po dokončení těchto klíčových milníků z důvodu omezených zdrojů a časových možností.

7.3 Doporučení

Zkušenosti získané při testování hry Preserve a při poskytování výpomoci v herním studiu byly využity k sepsání několika bodů, jejichž cílem je obohatit testování a vývoj budoucího herního projektu, inspirovat se přístupem studia nebo naopak poučit se z chyb, které byly při vývoji učiněny. Příklady budou prezentovány na projektu Preserve.

7.3.1 Rozdělit projekt na fáze

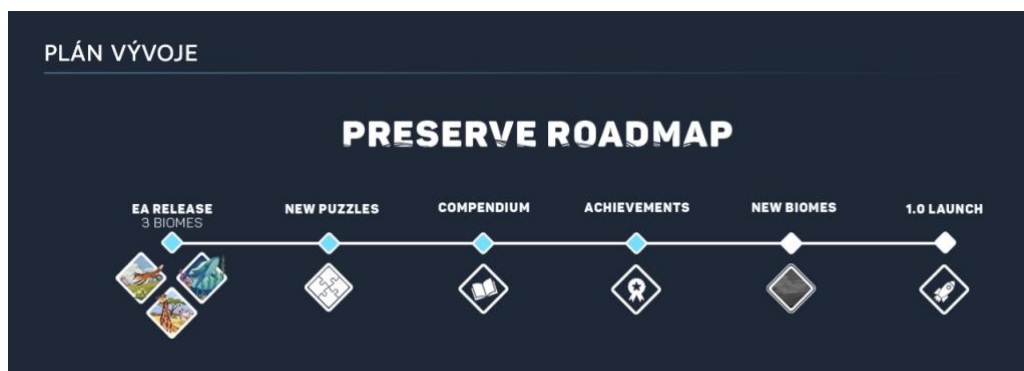
Herní projekt je rozčleněn na několik samostatných sekcí (grafická stránka, hudba a zvukové doprovody, příběh, optimalizace a lokalizace), které je nutné vzájemně koordinovat. Je považováno za náročné, aby byly všechny zmíněné aspekty současně řešeny menším týmem nebo jednotlivcem, a z tohoto důvodu byla kladena značná pozornost na detailní plánování a pečlivé stanovení priorit. U vývoje her srovnatelných s projektem Preserve bylo doporučeno nejprve vypracovat první plně hratelné demo a až poté do něj postupně implementovat další funkce a provádět opravy.

Pokud má být hra lokalizována do více jazyků nebo distribuována na různých platformách, doporučuje se definovat milníky a postupně je plnit. Při požadavku na podporu deseti jazyků se nejprve stanoví základní sada tří až čtyř nejpoužívanějších jazyků (například angličtina, slovenština, čeština a němčina), které jsou upřednostněny pro první vydání a jejich lokalizace je důkladně vyladěna. V následující aktualizaci jsou pak přidány další

čtyři jazyky, přičemž zajištění kvality těchto lokalizací je řešeno interně a detaily jsou po vydání doladěvány komunitou.

Stejný přístup může být uplatněn i při volbě cílových platform. Nejčastěji je hra vydávána prostřednictvím platformy Steam pro PC, jelikož je tento klient považován za dominantní. V případě, že je od počátku plánováno investování času do optimalizace pro více operačních systémů, bývá preferována podpora Linuxu (zejména kvůli SteamOS) a macOS. Testování na třech platformách bylo shledáno složitějším než testování na jedné, avšak díky statistikám z platformy Steam a provedené tržní rešerši je možné identifikovat platformu a hardwarová specifika s nejvyšší prioritou. Po vydání stabilní PC verze je doporučeno následně zaměřit se na implementaci podpory pro konzole. U konzolových verzí je nutno zohlednit licenční poplatky, akvizici vývojových kitů a schvalovací proces, který může trvat i několik dní. V souladu s tím by měla být podpora jednotlivých platform realizována postupně, a pokud je zamýšleno vydání na konzole již od počátku vývoje, měl by být použit multiplatformní herní engine a kód by měl být od počátku přizpůsobován pro různé cílové platformy. Tento model postupu je demonstrován praxí společnosti Rockstar Games, kde jsou hry nejprve uvedeny na konzolách PlayStation a Xbox a teprve po roce či dvou je následuje port pro PC.

Hráčům je prezentována „roadmapa“ vývoje, díky čemuž je jim umožněno vědět, co je čeká, a studiu vzniká veřejný závazek stanovené cíle naplnit. Doporučuje se soustředit testování a optimalizaci s dostatečnou časovou rezervou a v daném buildu či verzi již nepřidávat nové funkcionality. Obrázek 62 je uveden jako ilustrativní příklad. Pokud je v rámci větve (branche) plánováno uvedení nového biomu, je vyžadováno soustředit se výhradně na jeho vývoj, optimalizaci a odstranění chyb, bez přidávání dalších prvků (např. DLC či nových akčních karet), jež by zbytečně odváděly čas. V případě rozšíření o nový biom, zahrnujícího karty, pravidla, grafiku či hudbu, je doporučeno alokovat minimálně dvojnásobek času původně vymezeného pro vývoj.



Obrázek 62 Roadmapa hry Preserve

(zdroj: Stránka hry [70])

7.3.2 Budovat komunitu a „externí testery“

Hra je v menších studiích často uváděna do předběžného přístupu, aby byl získán první ohlas od hráčů. Získaný feedback je zaznamenáván formou návrhů na vylepšení a hlášení chyb, které jsou následně opraveny.

Na platformách jako Discord či Steam je komunikace s hráči vedena prostřednictvím diskusních vláken. V těchto kanálech jsou pro přehlednost hlášení využívány tagy nebo poznámky sloužící k označení nahlášených chyb. Po vyřešení je příslušné vlákno označeno jako uzavřené, aby byla zřejmá informace o odstranění daného problému.

Musí být zajištěno, aby byla komunikace vedena rychle, a hráčům bylo jasně sdělováno, že je o jejich podněty aktivně pečováno. Dále je doporučeno zavádět programy pro nejaktivnější testery/hráče, v nichž jsou přidělovány výhody jako přednostní přístup, odznaky v chatu či jiné pobídky, které mají zvýšit motivaci k dalšímu testování. Pro účely intenzivního odhalování chyb jsou hráči pravidelně zváni na cílené testování, při němž jim bývá poskytnut bonus výměnou za maximální „rozbíjení“ hry. Tímto způsobem je umožněno provádět playtesting a smoke testování, jimiž jsou odhaleny chyby, jež by jinak zůstaly neodhaleny.

7.3.3 Zaměřit přiměřený čas do automatizovaného testování

Bylo zjištěno v rámci testování Preserve nebo komunikace s NOXGames, že je automatizované testování využíváno pouze okrajově, přičemž převážná část týmu preferovala manuální testy z důvodu domnělého nižšího nákladu. Zároveň však bylo prokázáno, že integrační a unit testy mohou ušetřit čas již při prvotním vývoji komponent a zajistit jejich správnou funkčnost, zejména v oblasti regresního testování či ověřování

základní logiky. Je doporučeno, aby byly tyto testy integrovány do CI/CD pipeline a spouštěny po každém buildu, čímž by byla eliminována potřeba jejich manuální aktivace. V případě hry Preserve bylo uvedeno, že by se automatizované testy měly aplikovat především na zpracování konfiguračních souborů a na logiku bodovacího systému.

7.3.4 Využití umělé inteligence

Ve všech výše zmíněných doporučeních lze využít umělou inteligenci. V případě rozdělení projektu do fází, může AI (Artificial Intelligence) vygenerovat plán vývoje, kterého se držet, navrhnout, které jazyky a platformy vybrat prioritně, pomoci se samotným překladem nebo získat základní informace k portování her na konzole. Pro budování komunity lze použít moderní chatboty pro rychlejší komunikaci s hráči a následné třídění získaných informací. Pokud získá AI přístup do repositáře, je možné ji využít pro generování nových testů, optimalizaci těch stávajících nebo analýzu slabin.

V rámci testování toho AI nabízí více. Různé ML (Machine learning) modely dovedou ze získaných dat (na základě minulých chyb a frekvence změn v kódu) určit, které části hry testovat častěji nebo podrobněji. Ušetří se tím čas strávený analýzou a pokrytí všech oblastí. Využití ML-Agents pro Unity dokáže vytvořit jednoduchého „walker“ agenta, který umí procházet hrou, interagovat s objekty a detekovat pád hry.

ZÁVĚR

Byly navrženy, realizovány a vyhodnoceny testovací metodiky pro hru Preserve, nacházející se v předběžném přístupu. Nejprve byla vytvořena literární rešerše testovacích technik, na jejímž základě byly sestaveny testovací scénáře. Následně bylo podle těchto scénářů provedeno systematické testování hry a byly analyzovány získané výsledky. Na základě vyhodnocení byly formulovány doporučené postupy pro další vývoj a bylo posouzeno uplatnění jednotlivých metodik s ohledem na co nejlepší zkušenost hráče.

Dále bylo zjištěno, že všechny testovací techniky přinášejí přidanou hodnotu pro testování her a investice do nich se osvědčují. Méně časté techniky, jako zátěžové testy výdrže či testování přerušení, mohou doplnit základní rámec o krajní scénáře, avšak není nezbytné je zahrnovat do primárních testovacích scénářů. Automatizované testování nebylo identifikováno jako povinná součást vývojového procesu, avšak bylo konstatováno, že jeho částečné zavedení dokáže významně ušetřit čas a urychlit odstraňování chyb. Testování souladu s předpisy bylo považováno za méně kritické pro ne-konzolové tituly, nicméně bylo doporučeno jej využít k zajištění právní a ratingové shody před vydáním i distribucí.

Na základě analýzy výsledků testování bylo zjištěno, že testováním počtu nahlášených chyb hráči, zpětné vazby v recenzích a průběžného regresního testování došlo ke zlepšení stability hry i se současným rozšiřováním funkcionalit a přípravou portů na konzole. Bylo potvrzeno, že základní požadavky hráčů jsou naplňovány, přičemž zbývající chyby střední a nízké priority nebrání plynulému hernímu zážitku.

S vydáním verze 1.0 bylo doporučeno pokračovat s kontinuálním monitoringem veškerých chyb a jejich aktivní řešením. Předpokládá se, že rozšíření podpory na konzole učiní proces testování ještě více časově náročným. V důsledku toho bylo navrženo zvážit širší zavedení automatizovaných testů, navýšení kapacity QA týmu a využití AI nástrojů pro zefektivnění a zrychlení testovacích procesů.

V samostatném odstavci pro doporučení v rámci příštího vývoje bylo zmíněno využití umělé inteligence a jejich nástrojů pro snazší vývoj, rozdělení projektu do fází, aktivně pracovat s komunitou pro rozsáhlejší testování a zaměření se na automatizované testování. Budoucí práce mohou více nastínit využití AI v herním testování.

SEZNAM POUŽITÉ LITERATURY

- [1] What is Software Testing? Online. 2024. Dostupné z: <https://www.geeksforgeeks.org/software-testing-basics/>. [cit. 2025-02-03].
- [2] What is Software Testing? Definition, Types, and Tools. Online. 2025. Dostupné z: <https://katalon.com/resources-center/blog/software-testing>. [cit. 2025-02-03].
- [3] Testovací tým. Online. 2024. Dostupné z: <http://testovanisoftware.cz/manualni-testovani/testovaci-tym/>. [cit. 2025-02-16].
- [4] ISTQB [online]. Brusel: ISTQB not-for-profit association, 2022 [cit. 2022-11-06]. Dostupné z: <https://www.istqb.org>
- [5] SOFTWARE TESTING – BUG VS DEFECT VS ERROR VS FAULT VS FAILURE. Online. 2024. Dostupné z: <https://www.geeksforgeeks.org/software-testing-bug-vs-defect-vs-error-vs-fault-vs-failure/>. [cit. 2025-03-28].
- [6] ERROR, DEFECT, AND FAILURE. Online. 2022. Dostupné z: <https://www.toolsqa.com/software-testing/istqb/error-defect-failure/>. [cit. 2025-03-28].
- [7] Game Testing: All in One. 3rd ed. Dulles: Mercury Learning & Information, 2016. ISBN 9781942270768.
- [8] What Is the Software Testing Life Cycle? A Complete Guide. Online. 2025. Dostupné z: <https://www.testim.io/blog/software-testing-life-cycle/>. [cit. 2025-04-10].
- [9] A Detailed Guide on the Software Testing Life Cycle. Online. 2025. Dostupné z: <https://www.browserstack.com/guide/what-is-software-testing-lifecycle>. [cit. 2025-04-10].
- [10] PATTON, Ron. Testování softwaru. Praha: Computer Press, 2002. ISBN 80-7226-636-5.
- [11] What is Localization Testing? Explained With Best Practices. Online. 2025. Dostupné z: <https://testsigma.com/blog/localization-testing/>. [cit. 2025-04-06].
- [12] Manual Testing vs Automation Testing. Online. 2024. Dostupné z: <https://www.browserstack.com/guide/manual-vs-automated-testing-differences>. [cit. 2025-04-06].

- [13] Testing and quality assurance tips for Unity projects. Online. 2024. Dostupné z: <https://unity.com/how-to/testing-and-quality-assurance-tips-unity-projects#diagnostics-test-tooling>. [cit. 2025-04-04].
- [14] Unleashing Maximum Performance: Mastering Unity Game Testing for Optimal Results. Online. 2024. Dostupné z: <https://www.headspin.io/blog/mastering-unity-game-testing>. [cit. 2025-04-04].
- [15] Profiler introduction. Online. 2025. Dostupné z: <https://docs.unity3d.com/Manual/profiler-introduction.html>. [cit. 2025-04-04].
- [16] Functional Vs Non-Functional Testing: From A to Z. Online. <https://u-tor.com/topic/functional-vs-non-functional>. Dostupné z: 2023. [cit. 2025-03-17].
- [17] Functional Testing: A Detailed Guide. Online. <https://www.browserstack.com/guide/functional-testing>. Dostupné z: 2025. [cit. 2025-03-17].
- [18] What is Integration Testing? Definition, How-to, Examples. Online. 2025. Dostupné z: <https://katalon.com/resources-center/blog/integration-testing>. [cit. 2025-03-18].
- [19] *Just Say No to More End-to-End Tests*. Online. Dostupné z: <https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html>. [cit. -03-18].
- [20] Practical Unit Testing: 10 Strategies For Maximizing Test Efficiency. Online. 2023. Dostupné z: <https://wethinkapp.medium.com/practical-unit-testing-10-strategies-for-maximizing-test-efficiency-c0d3e64eec0e>. [cit. 2025-03-15].
- [21] Unit Testing. Online. 2025. Dostupné z: <https://tuskrr.app/learn/unit-testing>. [cit. 2025-03-15].
- [22] What is Unit Testing? Online. 2025. Dostupné z: <https://aws.amazon.com/what-is/unit-testing/>. [cit. 2025-03-15].
- [23] Integration Testing – Software Engineering. Online. 2024. Dostupné z: <https://www.geeksforgeeks.org/software-engineering-integration-testing/>. [cit. 2025-03-18].

- [24] What is Integration Testing? Definition, How-to, Examples. Online. 2025. Dostupné z: <https://katalon.com/resources-center/blog/integration-testing>. [cit. 2025-03-18].
- [25] What is End-to-End Testing? - A Complete Guide. Online. 2024. Dostupné z: <https://www.headspin.io/blog/what-is-end-to-end-testing#what-are-the-types-of-e2e-testing-methods>. [cit. 2025-03-19].
- [26] What is End To End Testing? Definition, Tools, Best Practices. Online. 2025. Dostupné z: <https://katalon.com/resources-center/blog/end-to-end-e2e-testing>. [cit. 2025-03-19].
- [27] End To End Testing: Tools, Types, & Best Practices. Online. 2024. Dostupné z: <https://www.browserstack.com/guide/end-to-end-testing>. [cit. 2025-03-19].
- [28] Best practices for localization testing. Online. 2025. Dostupné z: <https://www.globalapptesting.com/localization-testing>. [cit. 2025-03-11].
- [29] What is Localization Testing? Definition, How-To, and Use Cases. Online. 2024. Dostupné z: <https://testlio.com/blog/what-is-localization-testing/>. [cit. 2025-03-11].
- [30] What percentage of your software testing budget is allocated to regression testing? Online. 2022. Dostupné z: <https://www.gartner.com/peer-community/poll/percentage-software-testing-budget-allocated-to-regression-testing>. [cit. 2025-03-19].
- [31] What Is Regression Testing? Definition, Techniques, and Tools. Online. 2022. Dostupné z: <https://www.spiceworks.com/tech/devops/articles/what-is-regression-testing/>. [cit. 2025-03-19].
- [32] What is Regression Testing? Definition, Importance, and Process. Online. 2024. Dostupné z: <https://u-tor.com/topic/regression-testing-meaning>. [cit. 2025-03-19].
- [33] What is Non-Functional Testing? Online. <https://www.browserstack.com/guide/what-is-non-functional-testing>. Dostupné z: 2025. [cit. 2025-03-17].
- [34] What is Compliance Testing? How to do it? Online. 2023. Dostupné z: <https://www.testbytes.net/blog/compliance-testing/>. [cit. 2025-03-16].

- [35] Compliance Testing: Everything You Need to Know. Online. 2025. Dostupné z: <https://testsigma.com/blog/compliance-testing/>. [cit. 2025-03-16].
- [36] SECURITY TESTING – SOFTWARE TESTING. Online. 2024. Dostupné z: <https://www.geeksforgeeks.org/security-testing/#types-of-security-testing>. [cit. 2025-03-17].
- [37] SECURITY TESTING IN SOFTWARE TESTING. Online. 2024. Dostupné z: <https://www.practitest.com/resource-center/article/the-basics-of-security-testing/>. [cit. 2025-03-17].
- [38] SECURITY TESTING: 7 THINGS YOU SHOULD TEST, TOOLS AND BEST PRACTICES. Online. 2024. Dostupné z: <https://brightsec.com/blog/security-testing/>. [cit. 2025-03-17].
- [39] 14 Cybersecurity Metrics + KPIs You Must Track in 2025. Online. 2024. Dostupné z: <https://www.upguard.com/blog/cybersecurity-metrics>. [cit. 2025-03-17].
- [40] A Beginner's Guide to Usability Testing. Online. 2025. Dostupné z: <https://maze.co/guides/usability-testing/>. [cit. 2025-03-16].
- [41] Usability Testing – The Complete Guide. Online. 2024. Dostupné z: <https://www.uxtweak.com/usability-testing/>. [cit. 2025-03-16].
- [42] Advantages of Usability Testing and Some Drawbacks You Should Know. Online. 2021. Dostupné z: <https://u-tor.com/topic/advantages-of-usability-testing>. [cit. 2025-03-16].
- [43] What is Compatibility Testing? (Examples Included). Online. 2025. Dostupné z: <https://www.browserstack.com/guide/compatibility-testing>. [cit. 2025-03-13].
- [44] What is Compatibility Testing? Forward & Backward Example. Online. 2024. Dostupné z: <https://www.guru99.com/compatibility-testing.html#types-of-compatibility-tests>. [cit. 2025-03-13].
- [45] Performance testing, best practices, metrics & more. Online. 2024. Dostupné z: <https://www.tricentis.com/learn/performance-testing>. [cit. 2025-03-09].

- [46] What is Performance Testing? - A Comprehensive Guide. Online. 2025. Dostupné z: <https://www.headspin.io/blog/a-performance-testing-guide#types-of-performance-testing>. [cit. 2025-03-09].
- [47] What Is Gaming? Online. 2025. Dostupné z: <https://builtin.com/gaming>. [cit. 2025-04-12].
- [48] GDC State of Game Industry 2023. Online. 2023. Dostupné z: <https://www.scribd.com/document/625796642/Gdc-state-of-game-industry-2023>. [cit. 2025-04-13].
- [49] Certification Tested Xbox Requirements for Xbox Console Games. Online. 2023. Dostupné z: https://learn.microsoft.com/en-us/gaming/gdk/_content/gc/policies/console/console-certification-requirements-and-tests. [cit. 2025-04-13].
- [50] How to develop games for PlayStation: a beginner's guide. Online. 2024. Dostupné z: https://mainleaf.com/how-to-develop-games-for-playstation/#elementor-toc__heading-anchor-2. [cit. 2025-04-13].
- [51] What is cloud gaming? [online]. 2021 [cit. 2025-04-13]. Dostupné z: <https://www.digitaltrends.com/gaming/what-is-cloud-gaming-explained/>
- [52] How AR & VR in Gaming is Transforming Gameplay Experience? [online]. 2023 [cit. 2025-04-13]. Dostupné z: <https://300mind.studio/blog/ar-vr-in-gaming-industry/>
- [53] Motion Sickness. Online. 2022. Dostupné z: <https://wwwnc.cdc.gov/travel/page/motion-sickness>. [cit. 2025-04-13].
- [54] Key roles in a game development team – 2024 edition. Online. 2024. Dostupné z: <https://pinglestudio.com/blog/key-roles-in-a-game-development-team-2024-edition>. [cit. 2025-04-16].
- [55] Aleem, S., Capretz, L.F. & Ahmed, F. Game development software engineering process life cycle: a systematic review. *J Softw Eng Res Dev* 4, 6 (2016). <https://doi.org/10.1186/s40411-016-0032-7>
- [56] SOMMERVILLE, Ian. Software engineering. Online. 9th ed. Boston: Pearson, c2011. ISBN 978-0-13-703515-1. Dostupné

- z: <https://engineering.futureuniversity.com/BOOKS%20FOR%20IT/Software-Engineering-9th-Edition-by-Ian-Sommerville.pdf>. [cit. 2025-04-23].
- [57] RAMADAN, Rido a Yani WIDYANI. Game development life cycle guidelines. Indonesia: IEEE, 2013. ISBN 978-1-4799-4692-1
- [58] Agile vs. Waterfall: Which Development Methodology Works Best for Game Design? Online. 2024. Dostupné z: <https://www.linkedin.com/pulse/agile-vs-waterfall-which-development-methodology-works-a-sorrenti-eel9e>. [cit. 2025-04-23].
- [59] Průvodce Scrum. Online. 2020. Dostupné z: <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Czech.pdf>. [cit. 2025-04-23].
- [60] Spiral Model. Online. 2021. Dostupné z: <https://artoftesting.com/spiral-model>. [cit. 2025-04-23].
- [61] Spiral Model For Game Development: Techniques To Develop Games. Online. 2017. Dostupné z: <https://gamedevelopertips.com/spiral-model-for-game-development/>. [cit. 2025-04-23].
- [62] Combinatorial Testing | What it is, How to Perform & Tools. Online. 2023. Dostupné z: <https://testsigma.com/blog/combinatorial-testing/>. [cit. 2025-04-30].
- [63] Flowchart in Software Testing: A Comprehensive Guide with Examples. Online. 2023. Dostupné z: <https://qamind.com/blog/flowchart-in-software-testing-guide-with-examples/>. [cit. 2025-04-29].
- [64] Cleanroom Software Engineering Reference Model. Online. 1996. Dostupné z: https://insights.sei.cmu.edu/documents/1159/1996_005_001_16502.pdf. [cit. 2025-04-28].
- [65] Co jsou metriky incidentů MTBF, MTTR, MTTA a MTTF? Jak je vypočítat? Online. 2024. Dostupné z: <https://www.alvao.com/cs/blog/metriky-incidentu-mtbf-mttr-mtta-mttf-jak-je-vypocitat>. [cit. 2025-04-28].
- [66] 9 Different Types of Game Testing Techniques. Online. 2023. Dostupné z: <https://www.testbytes.net/blog/types-of-game-testing/>. [cit. 2025-04-28].

- [67] What is Game Testing and Its Various Types. Online. 2022. Dostupné z: <https://www.testscenario.com/types-of-game-testing/>. [cit. 2025-04-26].
- [68] What is Ad Hoc Testing: A Complete Guide. Online. 2025. Dostupné z: <https://www.lambdatest.com/learning-hub/ad-hoc-testing#skills-for-adhoc-testing>. [cit. 2025-05-18].
- [69] What is Playtesting, and Why Is It Important in Game Development? Online. 2024. Dostupné z: <https://www.linkedin.com/pulse/what-playtesting-why-important-game-development-kwalee-zb4uf>. [cit. 2025-05-18].
- [70] Preserve. Online. 2024. Dostupné z: <https://store.steampowered.com/app/2109270/Preserve/>. [cit. 2025-03-27].
- [71] Bitmap Galaxy. Online. 2025. Dostupné z: <https://bitmapgalaxy.com>. [cit. 2025-03-27].
- [72] Grindstone. Online. 2025. Dostupné z: <https://www.grindstone.sk>. [cit. 2025-03-27].
- [73] Videocard Benchmarks. Online. 2025. Dostupné z: <https://www.videocardbenchmark.net/compare/3805vs2825vs322vs2552/Intel-UHD-Graphics-620-vs-GeForce-GTX-750-vs-Radeon-HD-7770-vs-Intel-HD-5000>. [cit. 2025-02-16].
- [74] Preserve – Steam Charts. Online. 2025. Dostupné z: <https://steamdb.info/app/2109270/charts/#3m>. [cit. 2025-05-10].

SEZNAM OBRÁZKŮ

Obrázek 1 Vzájemný vztah mezi lidskou chybou, defektem a vadou systému	16
Obrázek 2 Typy testování	22
Obrázek 3 Test Runner okno v Unity	24
Obrázek 4 Testovací pyramida	25
Obrázek 5 Vodopádový model	42
Obrázek 6 Scrum metoda	43
Obrázek 7 Spirálový model.....	44
Obrázek 8 Porovnání lineárně herních specializovaných metod	46
Obrázek 9 Porovnání iterativních herních specializovaných metod	46
Obrázek 10 Zjednodušená reprezentace TVD	49
Obrázek 11 Příklad testovacího stromu	51
Obrázek 12 Ukázka ze hry	54
Obrázek 13 UI hry.....	55
Obrázek 14 Popis TS.....	59
Obrázek 15 Hlavička testovacího scénáře pro Tutoriál	60
Obrázek 16 Poznámky k testovacímu scénáři Tutoriálu.....	61
Obrázek 17 Tabulka s testovacími případy pro Tutoriál.....	61
Obrázek 18 Konec tutoriálu	62
Obrázek 19 Kombinatorická tabulka nastavení ovládání	63
Obrázek 20 Životní cyklus ticketu v Jiře	65
Obrázek 21 Ticket Jiry s nalezenou chybou	66
Obrázek 22 GE_TC010 – Chybné zobrazení v grafu vůči žebříčku.....	68
Obrázek 23 Chyba v TC010.....	69
Obrázek 24 GE_TC010 – Doplnění informace o nedostupnosti žebříčku.....	69
Obrázek 25 MSUI_TC011 – chyba v ovládání hlasitosti hudby	70
Obrázek 26 MSUI_TC023 – chyba v nastavení kvality textur	70
Obrázek 27 MSUI_TC024 – chyba ve velikosti rozlišení	71
Obrázek 28 MSUI_TC030 – chyba v ukládání hry	71
Obrázek 29 Chyba nekorektního zadávání názvů	71
Obrázek 30 MSUI_TC031 – chyba ve vytváření nové uložené pozice	72
Obrázek 31 CS_TC004 – Konflikt v přenastavení ovládání.....	73
Obrázek 32 TC006 –Nefunkční převrácení zoom	74
Obrázek 33 TC007 – Nefunkční vypnutí převrácení náklonu	74
Obrázek 34 Kombinatorická tabulka ovládání s výsledky.....	74

Obrázek 35 Herní strom.....	75
Obrázek 36 Tabulka terminologie	76
Obrázek 37 Správné formátování data napříč jazyky	76
Obrázek 38 I. příklad lokalizační chyby	78
Obrázek 39 II. příklad lokalizační chyby.....	79
Obrázek 40 III. příklad lokalizační chyby	80
Obrázek 41 Výkonnostní porovnání GPU z tabulek a vlastněných autorem	82
Obrázek 42 Minimální požadavky – HW – Windows.....	83
Obrázek 43 Grafické glitche a artefakty při nízkém rozlišení	84
Obrázek 44 Vytíženost PC.....	84
Obrázek 45 Minimální požadavky – Mac.....	85
Obrázek 46 Doporučené požadavky – Mac	86
Obrázek 47 Vytíženost Mac	87
Obrázek 48 Vytíženost Linux	88
Obrázek 49 Využití CPU na střední detaily.....	89
Obrázek 50 Využití paměti na střední detaily	89
Obrázek 51 Graf výdrže baterie – Výkon.....	92
Obrázek 52 Graf výdrže baterie – Výdrž.....	93
Obrázek 53 Porovnání FPS před a po změně velikosti okna hry.....	96
Obrázek 54 Steam Deck kompatibilita	97
Obrázek 55 Steam diskuse	105
Obrázek 56 Discord kanál pro hlášení chyb	106
Obrázek 57 Discord – chyba propisování vodních dlaždic z minulé hry	106
Obrázek 58 Graf počtu chyb podle kategorií.....	120
Obrázek 59 Graf počtu bugů podle kategorie a priority	121
Obrázek 60 Graf stavu chyb ve verzi 0.9.16.....	122
Obrázek 61 Steam statistiky hry Preserve	123
Obrázek 62 Roadmapa hry Preserve.....	127

SEZNAM TABULEK

Tabulka 1 Bodovací tabulka	56
Tabulka 2 HW a SW požadavky Windows	83
Tabulka 3 HW a SW požadavky Mac.....	85
Tabulka 4 Steam/Linux.....	87
Tabulka 5 Vytížení při režimu Výkon	91
Tabulka 6 Vytížení při režimu Výdrž	93
Tabulka 7 Tabulka rozložení dle priorit.....	120

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ACIP	Age classification of information products
ACTS	Advanced Combinatorial Testing System
AD	Assembly definition
AMD	Advanced Micro Device
AI	Artificial Intelligence
AR	Augmented Reality
ARM	Advanced RISC Machine
CADP	A China Audio-video and Digital Publishing Association
CERO	Computer Entertainment Rating Organization
CPU	Central Processing Unit
CSRF	Cross-Site Request Forgery
E2E	End-to-End (testování)
ESRB	Entertainment Software Rating Board
FPS	Frames Per Seconds
GDC	Game Developers Conference
GDLC	Game Development Life Cycle
GDPR	General Data Protection Regulation
GLSL	OpenGL Shading Language
GRAC	Game Rating and Administration Committee
GUI	Graphical User Interface
HDD	Hard Disk Drive
HIPAA	Health Insurance Portability and Accountability Act
HLSL	High-Level Shading Language
HUD	Head-up display
HW	Hardware
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
KPI	Key Performance Indicator
ML	Machine learning
MTTD	Mean time to detect
MTTF	Mean time to failure

MTTR	Mean time to repair
OBS	Open Broadcaster Software (OBS Studio)
PEGI	Pan European Game Information
QA	Quality Assurance
RIS	Radeon Image Sharpening
RISC	Reduced Instruction Set Computer
SDK	Software development kit
SDLC	Software Development Life Cycle
SoC	System on a chip
SOC2	Service Organization Control 2
SQL	Structured Query Language
SSD	Solid State Drive
STLC	Software Testing Life Cycle
SW	Software
TC	Testovací případ (Test case)
TRC	Technical Requirements Checklist
TS	Testovací scénář (Test scenario)
UI	User Interface (Uživatelské prostředí)
USK	Unterhaltungssoftware Selbstkontrolle
UTF	Unity Test Framework
UX	User experience
VR	Virtual Reality
W3C	World Wide Web Consortium
XSS	Cross-Site Scripting
XR	Xbox Requirements

SEZNAM PŘÍLOH

Příloha P I: Složka s prací

PŘÍLOHA P I: SLOŽKA S PRACÍ

- Diplomová práce ve formátu .pdf
- Soubor *Preserve* ve formátu .zip
 - a. Složka *Assets/Tests* – složka s automatizovanými testy
- Složka *Testovací scénáře*
 - a. Složka *Functional* – Složka s funkcionálními testy
 - i. *Continental – Puzzles.xlsx*
 - ii. *Marine – Puzzles.xlsx*
 - iii. *Savana – Puzzles.xlsx*
 - iv. *TS – Functional tests.xlsx*
 - b. *TS – Compatibility.xlsx*
 - c. *TS – Compliance.xlsx*
 - d. *TS – Controller.xlsx*
 - e. *TS – Localisation.xlsx*
 - f. *TS - Performance.xlsx*
- Složka *Compendium* – složka s encyklopedií herních biomů
 - a. *Kontinentální biom.pdf*
 - b. *Mořský biom.pdf*
 - c. *Savana biom.pdf*
- Složka *Výsledky testování*
 - a. *Bug report.xlsx* – seznam všech chyb včetně jejich stavu a různých statistik
 - b. *Preserve_Linux.xlsx* – Tabulka zachycující HW vytížení hry na Linuxu
 - c. Složka *Lokalizační testování* – složka obsahuje záznamy různých chyb
 - d. Složka *Testování kompatibility* – složka obsahuje záznamy různých chyb
 - e. Složka *Výkonnostní testování* – složka obsahuje záznamy různých chyb