Testování v herním průmyslu

Bc. Zdeněk Pohludka

Diplomová práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

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

1. Vypracujte literární rešerši na zadané téma.
2. Analyzujte jednotlivé metody testování používané ve vývoji počítačových her.
3. Navrhněte testovací scénáře a způsoby testování pro vybranou hru ve vývoji.
4. Realizujte testování s využitím vhodných technik.
5. Analyzujte 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: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

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

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

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomové práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky. Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že 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;
- beru na vědomí, že 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;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že 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 samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne .....Zdeněk Pohludka v. r.....

podpis studenta

ABSTRAKT

Diplomová práce se zaměřuje na problematiku testování v herním průmyslu. Teoretická část se věnuje softwarovému testování, hernímu průmyslu a specifickým aspektům herního testování. Následuje analýza životního cyklu herního vývoje, typy defektů a testovací techniky. Praktická část analyzuje testovací procesy ve společnosti NOXGames a SCS Software s.r.o. a implementuje různé testovací metodiky do vlastního projektu i známých her, včetně kombinatorického testování, testovacích vývojových diagramů a cleanroom testování. Závěrečná analýza řešení shrnuje poznatky z práce a vysvětluje která řešení se hodí do vybraného projektu a která nikoli.

Klíčová slova: testování, herní průmysl, software, programování, Unity (software)

ABSTRACT

The thesis focuses on the issue of testing in the gaming industry. The theoretical part is devoted to software testing, game industry and specific aspects of game testing. This is followed by an analysis of the game development life cycle, types of defects and testing techniques. The practical part analyses the testing processes at NOXGames and SCS Software Ltd. and implements different testing methodologies in own projects and well-known games, including combinatorial testing, test flowcharts and cleanroom testing. The final solution analysis summarizes the findings of the work and explains which solutions fit the selected project and which do not.

Keywords: testing, game industry, software, programming, Unity (software)

Tímto bych rád poděkoval vedoucímu práce Ing. Tomáši Vogeltanzovi Ph.D. za zpětnou vazbu a rady. Dominiku Mačenkovi a hernímu studiu NOXGames, kteří mi poskytli informace o jejich testovacích procesech. Lukáši Petrovi, který mi pomáhal během práce komunikovat s SCS Software s.r.o., Kristýně Routnerové a studiu samotnému, kteří mi poskytli informace o testování. A na závěr mé ženě Anně, která mi byla oporou v celé době studia.

OBSAH

ÚVOD ... **10**

I TEORETICKÁ ČÁST ... **11**

1 STRUČNÁ HISTORIE ... **12**

2 SOFTWAROVÉ TESTOVÁNÍ ... **14**

2.1 Typy testování ... 14

2.2 Manuální a automatizované testování ... 14

2.3 Funkcionální testy ... 15

2.3.1 Unit testování ... 16

2.3.2 Integrační testy ... 17

2.3.3 Systémové testování ... 18

2.3.4 Akceptační testování ... 19

2.4 Nefunkcionální testování ... 20

2.4.1 Bezpečnostní testování ... 20

2.4.2 Výkonostní testování ... 21

2.4.3 Testování použitelnosti ... 21

2.5 Zbylé oblasti ... 22

3 HERNÍ PRŮMYSL ... **24**

3.1 Herní platformy ... 24

3.1.1 PC ... 24

3.1.2 Konzole ... 24

3.1.3 Cloud ... 25

3.1.4 VR/AR ... 25

3.1.5 Mobilní platformy ... 25

3.1.6 Zhodnocení rozdílů platforem z hlediska testování a kontroly kvality: 26

3.2 Vývoj v herním průmyslu ... 27

3.2.1 Waterfall ... 27

3.2.2 Agile ... 27

3.2.3 Jednotný proces vývoje ... 28

4 HERNÍ TESTOVÁNÍ ... **29**

4.1 Životní cyklus herního vývoje ... 30

4.1.1 Životní cyklus softwarového vývoje ... 30

4.1.2 Blitz Games Studios GDLC ... 30

4.1.3 Arnold Hendrick's GDLC ... 31

4.1.4 Doppler Interactive GDLC ... 33

4.2 Typy defektů ... 34
4.2.1 Funkce ... 35
4.2.2 Přiřazení ... 35
4.2.3 Kontrola ... 36
4.2.4 Načasování ... 37
4.2.5 Build chyby ... 37
4.2.6 Algoritmus ... 39
4.2.7 Dokumentace ... 40
4.2.8 Interface ... 41
4.3 Testovací techniky ... 42
4.3.1 Kombinatorické testování ... 42
4.3.2 Testovací vývojové diagramy ... 49
4.3.3 Cleanroom testování ... 61
4.3.4 Testovací stromy ... 67
4.3.5 Play a Ad Hoc testování ... 71
II PRAKTICKÁ ČÁST ... 75
5 ANALÝZA TESTOVACÍHO PROCESU: POHLED Z PRAXE ... 76
5.1 NOXGames ... 76
5.1.1 Dotazník ... 76
5.1.2 Souhrn ... 78
5.2 SCS Software s.r.o ... 78
5.2.1 Dotazník ... 78
5.2.2 Souhrn ... 80
6 TESTOVACÍ METODIKY ... 81
6.1 Kombinatorické testování ... 81
6.2 Testovací vývojové diagramy ... 86
6.2.1 Vytvoření vývojového diagramu ... 86
6.2.2 Vytvoření datového slovníku ... 88
6.2.3 Vytvoření cest ... 89
6.2.4 Diagramy pro smrt hráče a chování mince ... 90
6.3 Cleanroom testování ... 94
7 FUNKCIONÁLNÍ TESTOVÁNÍ ... 99
7.1 Unit testování ... 102
7.1.1 TimerTests testy ... 102
7.1.2 EnemyMovementTests testy ... 106
8 NEFUNKCIONÁLNÍ TESTOVÁNÍ ... 109

8.1 Výkonnostní testování 109
8.2 Testování použitelnosti 111
8.2.1 Příprava a stanovení cílů 112
8.2.2 Výběr vhodných hráčů 113
8.2.3 Průběh testování 114
8.2.4 Analýza a zpracování výsledků 114
9 ANALÝZA ŘEŠENÍ 115
9.1 Kombinatorické testování 115
9.2 Testovací vývojové diagramy 116
9.3 Cleanroom testování 117
9.4 Funkcionální testování 117
9.5 Unit testování 118
9.6 Výkonnostní testování 118
9.7 Testování použitelnosti 119
ZÁVĚR 120
SEZNAM POUŽITÉ LITERATURY 122
SEZNAM POUŽITÉ LITERATURY 126
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK 126
SEZNAM OBRÁZKŮ 127
SEZNAM TABULEK 128

ÚVOD

Herní průmysl se v posledních letech význačně rozrost a v blízké budoucnosti by nic nemělo tomuto růstu bránit. Ať už velké AAA studia, tak i malí indie vývojáři vyvíjí hry, které se dokáží dostat do celého světa. Téma testování v herním průmyslu je stále aktuální a relevantní, neboť s rostoucí složitostí a ambicemi vývojářských týmů se zvyšuje i potřeba zajistit vysokou kvalitu, spolehlivost a uživatelskou spokojenost s herními produkty. Vývoj se stal dostupnější než kdykoliv předtím, ale bohužel to nezaručilo bezchybnost her.

Tato diplomová práce si klade za cíl prozkoumat a analyzovat problematiku testování v herním průmyslu z teoretického i praktického hlediska. V současné době existuje mnoho různých metodik a přístupů k testování softwaru ve hrách, avšak jejich efektivita a vhodnost pro konkrétní situace mohou být často nejasné. Právě proto je důležité provést studii, která by objasnila nejen samotné procesy testování, ale i jejich aplikaci a účinnost v praxi.

V rámci této práce bude představena krátce historie testování, budou analyzovány nejen základní principy softwarového testování, ale i specifika herního průmyslu, včetně rozdílů v testovacích strategiích mezi jednotlivými herními platformami a vývojovými metodologiemi. Dále se práce zaměří na konkrétní aplikaci testovacích metodik ve společnostech NOXGames a SCS Software s.r.o., které slouží jako příklad vývojových studií v herních odvětví. Studio NOXGames se zaměřuje na mobilní hry a v jejich portfoliu se nachází různé žánry od strategie, po závodní hry až po karetní. Naopak SCS Software s.r.o. se soustředí na konzole a desktopové platformy, kdy jejich primárním žánrem jsou simulátory. Pracovní hypotézy budou formulovány na základě analýzy současného stavu testování ve hrách a očekávaného přínosu jednotlivých přístupů pro zlepšení kvality a spolehlivosti herních produktů.

Očekávaný přínos této práce spočívá v poskytnutí užitečných poznatků a doporučení pro malé vývojářské týmy v herním průmyslu, které by jim pomohly optimalizovat jejich testovací procesy a dosáhnout lepších výsledků při vydávání nových her na trh. Dalším přínosem je prohloubení autorovy znalosti o testování softwaru ve hrách a o vývoji herních produktů. Zároveň by tato práce mohla sloužit jako zdroj inspirace pro budoucí výzkum v oblasti testování softwaru ve hrách.

Autor této práce se živí testováním softwaru webových aplikací a má přes dva roky zkušeností. Testuje jak manuálně (primárně funkcionální a optimalizační testování), tak i automatizovaně za pomoci Cypress nebo Postmanu. rád vyzkoušel práci v herním průmyslu. Toto je jeho osobní důvod k vytvoření této práce, která mu má umožnit představit možné metodiky pro testování her a vyzkoušet si je na vlastním projektu, který sám vyvíjí.

I. TEORETICKÁ ČÁST

1 STRUČNÁ HISTORIE

Na počátku padesátých let nebyl rozdíl mezi testováním a laděním softwaru. Vývojáři při tvorbě aplikací museli v případě nalezení chyby chybu jednoduše zanalyzovat a opravit. Žádný koncept testování tehdy neexistoval. Vše se změnilo v roce 1957, kdy Charles L. Baker napsal recenzi na knihu Digital Computer Programming od Dana McCrackena, kdy definoval rozdíly mezi testováním a laděním. [6, 8]

Mezi léty 1957 až 1978 bylo rozlišováno mezi dvěma hlavními činnostmi – laděním a testováním softwaru. Testování bylo v té době samostatnou činností, zaměřenou na zajištění splnění požadavků na software. Testeři se zaměřovali na ověření konkrétních požadavků, jako například omezení na zobrazení pouze 10 produktů ve webové aplikaci. V tomto období nebyl běžný koncept negativního testování, tj. záměrného rozbití aplikace, a testování se primárně zaměřovalo na ověření správné funkcionality softwaru. S nárůstem počtu testů však rostla i pravděpodobnost nalezení chyb, a proto bylo třeba hledat nové přístupy k zajištění kvality softwaru. [6, 8]

V letech 1979 až 1982 se zaměření na testování soustředilo na prolomení kódu a odhalení chyb v něm. Leč Glenford J. Myers hrál klíčovou roli, když v roce 1979 oddělil ladění od testování, jeho pozornost byla zaměřena na testování zlomů. Jeho myšlenkou bylo, že úspěšný testovací případ odhalí dosud neobjevenou chybu. Toto období se vyznačovalo snahou oddělit základní vývojové činnosti, jako je ladění, od testování. Testování v té době zahrnovalo i testování destruktivními metodami, které ale nedokázalo preventivně předejít chybám. Tento přístup byl výzvou, protože odhalené chyby byly obtížné opravit bez vytváření nových. [6, 8]

V období let 1983 až 1987 se důraz ve sféře testování přesunul k hodnocení a měření kvality softwaru. Testování hrálo klíčovou roli v zvyšování důvěry ve funkčnost softwaru. Zaměřováním se na systematické testování a hodnocení softwaru s cílem dosáhnout přijatelné úrovně, s počtem minimálních chyb byl zejména významný princip při testování rozsáhlého softwaru. Cílem bylo zajistit, že software dosáhne stanovené úrovně kvality a důvěryhodnosti prostřednictvím opakovaných a systematických testovacích cyklů. [6, 8]

V období let 1988 až 2000 došlo k novému přístupu v testování softwaru, kde se důraz klade na ověření, zda software splňuje specifikace, odhalení chyb a prevenci vad. Kód byl rozdělen na testovatelný a netestovatelný, přičemž testovatelný kód byl považován za méně chybový než kód obtížně testovatelný. Během této éry bylo klíčové rozvíjet techniky testování. Začalo se zdůrazňovat průzkumné testování, kde tester systematičtěji prozkoumával a hlouběji porozuměl softwaru s cílem identifikovat více chyb. [6, 8]

Nakonec po roce 2000 přineslo testování softwaru několik inovativních konceptů, včetně Vývoje řízeného testy (TDD) a Vývoje řízeného chováním (BDD), které zdůrazňují integraci testování do vývojového procesu. V roce 2004 došlo k zásadnímu posunu v oblasti testování s příchodem automatizovaných testovacích nástrojů, kde lze zmínit například Selenium. Tato epocha přinesla efektivnější testování softwaru, a zároveň se prosadilo testování rozhraní API pomocí nástrojů jako SOAP UI. V současné éře směřuje testování k možnostem využití nástrojů umělé inteligence (AI) a testování napříč prohlížeči s pomocí nástrojů jako SauceLabs a Browserstack. Tabulka 1.1 odkazuje na letopočty s milníky testování. [6, 8]

Tabulka 1.1 Historické milníky testování

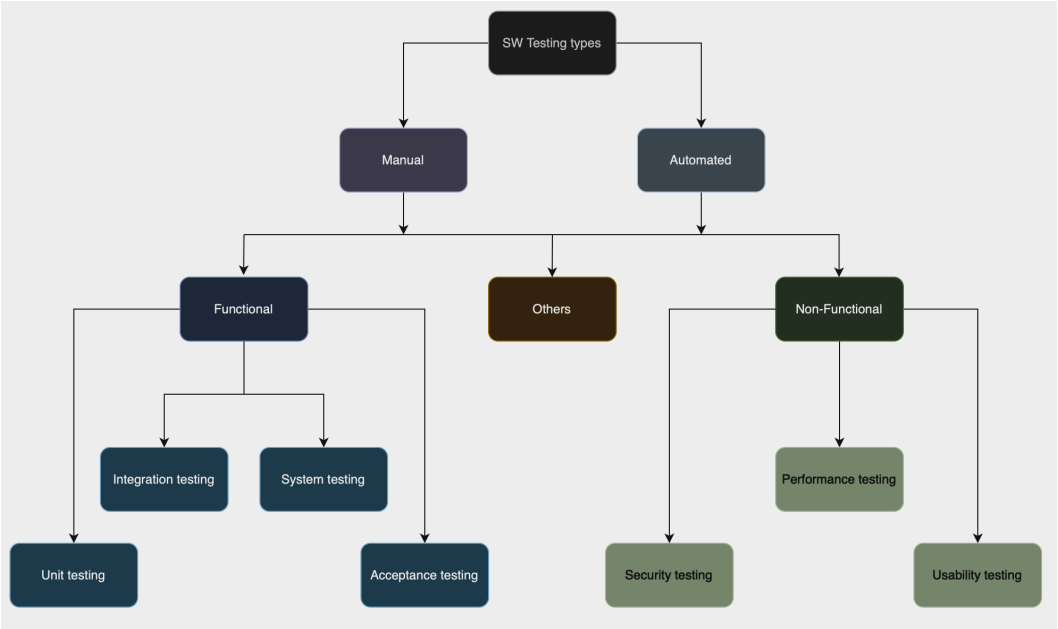
do roku 1957	neexistovalo
1958-1978	testování a ladění
1979-1982	uspokojení požadavků
1983-1987	prolamování kódu
1988-2000	měření kvality
2001-současnost	AI, cross-browser testing, atp.

2 SOFTWAROVÉ TESTOVÁNÍ

Testování softwaru je nedílnou součástí vývoje aplikace. V případě dodání nekompletní nebo vadné aplikace by mohly být následky vážné a problematické jak pro klienta, tak pro vývojáře. Pokud má společnost dobře nastavené techniky testování a QA procesy, může firma ušetřit nemalé peníze. [1, 7]

2.1 Typy testování

Validace základních požadavků a testování jsou klíčovými aspekty hodnocení softwaru. Validace se zaměřuje na kritické posouzení základních požadavků, zatímco průzkumné testování pomáhá odhalit nečekané scénáře a situace, které mohou vést k chybám. Dokonce i jednoduché aplikace mohou vyžadovat širokou škálu různorodých testů. Efektivita testování je optimalizována minimalizací počtu testů potřebných k odhalení co největšího množství defektů. Základní rozdělení typů testování je na Obrázku 2.1. [1]



Obrázek 2.1 Rozdělení testovacích typů

2.2 Manuální a automatizované testování

Manuální testování vykonává jednotlivec, který interaguje s aplikací nebo rozhraním API pomocí příslušných nástrojů. Mezi možné nevýhody může být fakt, že testy obvykle mají nižší rychlost provedení. Na druhou stranu jistým benefitem je i skutečnost, že pro manuální testování lze získat i člověka s nižší kvalifikací (znalostmi). Tento přístup bývá nákladný, protože vyžaduje manuální nastavení prostředí a provedení testů, což může být náchylné k lidským chybám, jako jsou překlepy nebo opomenuté kroky v

testovacím skriptu. [1, 9]

V případě automatizovaného testování je proces proveden strojem, který vykonává předem napsaný testovací skript. Složitost těchto testů může zahrnovat kontrolu jednotlivých metod ve třídě až po ověření, že provedení sekvence složitých akcí v uživatelském rozhraní vede ke stejným výsledkům. Problém může nastat v případě testování UI. Testy mohou být křehké na změny v rozložení prvků na stránce. Stačí, aby se snížilo rozlišení a test, který byl předtím úspěšný, může skončit chybou.

Automatizované testy jsou mnohem odolnější a spolehlivější než ty manuální, přičemž kvalita automatizovaných testů závisí na dovednostech napsání příslušných testovacích skriptů. [1, 9]

2.3 Funkcionální testy

Funkční testování je forma testování, která se zaměřuje na ověření, zda funkce zkoumaného systému pracují tak, jak je očekáváno a v souladu s definovanými požadavky. Hlavním cílem tohoto typu testování je zabezpečit, aby software plnil své stanovené funkce, aniž by se podrobně zabýval jeho interní strukturou kódu nebo implementačními detaily. Funkční testování je považováno za jeden z nejběžnějších a základních typů testování. Když lidé hovoří o "testování"v obecném smyslu, často tím míní právě funkční testování. [7, 12, 9]

Následující testovací scénář slouží k ilustraci toho, jak můžou funkcionální testy být sepsány. Testovací scénáře mají za cíl prověřit, že daný software funguje podle specifikace a byly splněny všechny požadavky.

Scénář: Age of Empires: "Ascent of Egypt - Foraging"

Cíl scénáře: Ověřit, zda hráč úspěšně dokončí v kampani misi "Foraging"ve hře Age of Empires (rok vydání 1997). To zahrnuje postavit storage pit (skladiště), granary (sýpka), dock (přístav) a rozšířit svou populaci na 12 pracovníků.

1. Spustit kampaň "Foraging".
2. Ověřit, že se zobrazí informační panel s historií, pokyny a nápovědou.
3. Po kliknutí na OK se zobrazí hra, kde jsou 3 pracovníci a hlavní chýše.
4. Těžba surovin:
   (a) Ověřit, že hráč může těžit dostatek jídla a dřeva.
   (b) Zkontrolovat, že pokud pracovnici vytěží 10 jednotek dané suroviny, tak po přinesení do chýše se zvýší počet této suroviny.
5. Stavba Storage Pit (Skladiště)/Stavba Granary (Sýpka)/Stavba Dock (Přístav):

(a) Ujistěte se, že pro postavení budovy je dostatek surovin.

(b) Pokud je dostatek surovin a vybere se jeden pracovník, je možné vybrat budovu a postavit ji.

(c) Ověřte, že hráč může postavit danou budovu.

6. Rozšíření populace na 12 pracovníků:

(a) Ujistěte se, že pro vytvoření pracovníka je dostatek surovin a domečků.

(b) Pokud je dostatek surovin a vybere se chýše, je možné vybrat pracovníka a vytvořit ho.

(c) Ověřte, že hráč může vytvořit pracovníka.

7. Dokončení kampaně:

(a) Ověřte, že hráč dokončil misi úspěšně podle stanovených cílů.

(b) Jakmile se splní cíle, zazní fanfára a zobrazí se text "You are victorious!".

(c) Zobrazí se stránka se statistikami.

2.3.1 Unit testování

- White box testování

Unit testy se zaměřují na velmi nízkou úroveň a přímo vstupují do zdrojového kódu aplikace. Provádějí testování jednotlivých metod a funkcí tříd, komponent nebo modulů používaných v softwaru. Tyto testy jsou obecně cenově dostupné pro automatizaci a lze je rychle spouštět pomocí serveru kontinuální integrace. Hlavním účelem serveru kontinuální integrace je automatizovat procesy jako sestavení aplikace, spouštění testů a nasazení do vývojového, testovacího nebo produkčního prostředí. [7, 10, 9]

Příklad implementace: [10]

```
#include <iostream>
#include <cassert>

// Function for adding two numbers
int add(int a, int b) {
    return a + b;
}

// Class for unit tests
class TestAddFunction {
public:
    // Test case 1
    void test_add_1() {
        assert(add(5, 10) == 15);
    }

    // Test case 2
    void test_add_2() {
        assert(add(500, 1000) == 1500);
    }

    // Test case 3
    void test_add_3() {
```

```
27              assert(add(0, 1000) == 1000);
28          }
29
30          // Test case 4
31          void test_add_4() {
32              assert(add(-100, 100) == 0);
33          }
34
35          // Test case 5
36          void test_add_5() {
37              assert(add(-100, -1100) == -1200);
38          }
39      };
40
41      int main() {
42          TestAddFunction test;
43
44          // Run the test cases
45          test.test_add_1();
46          test.test_add_2();
47          test.test_add_3();
48          test.test_add_4();
49          test.test_add_5();
50
51          std::cout << "All tests passed!" << std::endl;
52
53          return 0;
54      }
```

2.3.2 Integrační testy

- Gray box testování

Obvyklý softwarový projekt se zpravidla skládá z mnoha softwarových modulů, z nichž mnohé vytvořili různí programátoři. Integrační testování ukazuje týmu, jak dobře tyto různorodé prvky spolupracují. Slouží k ověření správné spolupráce různých modulů nebo služeb, které jsou využívány v aplikaci. Někdy dochází k záměně integračních a funkčních testů, protože oba vyžadují vzájemnou interakci více komponent. Rozdíl spočívá například v tom, že integrační test může jednoduše ověřit, že se lze dotazovat do databáze, zatímco funkční test by očekával získání konkrétní hodnoty z databáze, jak je definováno v požadavcích na produkt. Tyto testy mohou zahrnovat kontrolu, zda mikroslužby pracují společně tak, jak je očekáváno. Jejich provedení je náročnější, protože vyžadují provoz více částí aplikace a často zahrnují simulaci reálného prostředí. [7, 11, 9]

Tento scénář poskytuje strukturovaný přístup k integračnímu testování multiplayerového systému v hře Call of Duty a zahrnuje testování komunikace mezi hráči a synchronizaci herního stavu mezi klienty a herním serverem.

Scénář: Call of Duty - Komunikace multiplayerového systému

1. Spuštění hry a připojení k multiplayerovému serveru:
   (a) Připojení k serveru a zobrazení seznamu dostupných her.
   (b) Výběr hry a vstup do herní místnosti.
2. Hraní multiplayerové hry a komunikace mezi hráči:

   (a) Odeslání chatové zprávy v herní místnosti.

   (b) Přijetí chatové zprávy od jiného hráče.

   (c) Odeslání pozvánky ke spolupráci nebo souboji jinému hráči.

3. Synchronizace herního stavu mezi hráči:

   (a) Aktualizace polohy a stavu hráče u všech ostatních hráčů.

   (b) Synchronizace událostí jako střelba, zásahy nebo změny prostředí.

- Testování komunikace mezi herními servery a klienty:

  1. Ověření, že klienti se mohou správně připojit k hernímu serveru a synchronizovat herní stav.
  2. Testování odesílání a přijímání chatových zpráv mezi hráči v reálném čase.

- Testování synchronizace herního stavu:

  1. Ověření, že polohy a akce hráčů jsou správně synchronizovány mezi všemi hráči v herní místnosti.
  2. Testování reakce na události, jako jsou střelby nebo zásahy, a zajištění, že jsou správně zpracovány všemi klienty.

2.3.3 Systémové testování

- Black box testování
- End-to-end testování
- Smoke testování

Je fáze ověřování, která se zaměřuje na to, zda aplikace plní své úkoly podle zamýšleného návrhu. Tato forma Black boxu klade důraz na funkčnost aplikace a nevěnuje se příliš interním operacím systému, což je oblastí testování White boxu. Během systémového testování se zkoumá, zda různé typy uživatelských vstupů vedou k požadovaným výstupům v celém rozsahu aplikace. Tato fáze testování se obvykle provádí před Akceptačním testováním a po Integračním testování.

Příklad zdrojového kódu pro systémové testování. Využívá cURL knihovnu z C++, která provádí HTTP požadavky a testuje konektivitu a odpověď z určitému endpointu. [15]

```cpp
#include <iostream>
#include <string>
#include <curl/curl.h>
#include <cassert>

// Callback function for handling HTTP request response
size_t writeCallback(void *contents, size_t size, size_t nmemb, std::string *data) {...}

// Function for executing GET request
std::string get_request(const std::string& url) {...}

// Function for executing POST request
std::string post_request(const std::string& url, const std::string& data) {...}

// Class for tests
class SystemTests {
public:
    // Test for GET request for users
    void test_get_users(const std::string& url) {
        std::string response_data = get_request(url + "/users");
        assert(response_data != "");
        // Here you can perform verification of the response content, such as JSON parsing and content check
        std::cout << "GET Users Test Passed!" << std::endl;
    }

    // Test for POST request for users
    void test_post_users(const std::string& url) {
        std::string user_data = "{\"name\": \"John\", \"email\": \"john@example.com\"}";
        std::string response_data = post_request(url + "/users", user_data);
        assert(response_data != "");
        // Here you can perform verification of the response content, such as JSON parsing and content check
        std::cout << "POST Users Test Passed!" << std::endl;
    }
};

int main() {
    std::string api_url = "http://your_api_url";
    SystemTests tests;

    // Running the tests
    tests.test_get_users(api_url);
    tests.test_post_users(api_url);

    return 0;
}
```

2.3.4 Akceptační testování

- Alfa testování
- Beta testování

Akceptační testování je proces testování černé skříňky, kde se hodnotí funkčnost softwaru s cílem zajistit, že produkt splňuje stanovená kritéria přijatelnosti. Tento přístup k testování se zaměřuje na ověření přijatelnosti celého systému. Akceptační testování tvoří poslední fázi celého procesu testování softwaru a má klíčový význam před uvedením softwaru do skutečného provozu. Po dokončení testování systému, opravě většiny chyb a jejich ověření a uzavření je čas přejít k akceptačnímu testování. [7, 14, 9]

2.4 Nefunkcionální testování

Slouží k hodnocení výkonnosti, použitelnosti, spolehlivosti a dalších nefunkčních vlastností softwarové aplikace. Tento typ testování je zaměřen na posouzení připravenosti systému z hlediska nefunkčních kritérií, která běžné funkční testování nezahrnuje. Je nezbytným krokem pro potvrzení spolehlivosti a funkčnosti softwaru. Jeho základem jsou specifikace požadavků na software, které umožňují týmům zajišťujícím kvalitu ověřit, zda systém plně splňuje požadavky uživatelů. Cílem nefunkčního testování je zlepšení použitelnosti, efektivity, udržovatelnosti a přenositelnosti produktu. Tímto způsobem pomáhá minimalizovat výrobní rizika spojená s nefunkčními aspekty produktu. [13]

2.4.1 Bezpečnostní testování

Hlavním úkolem je identifikovat a řešit bezpečnostní nedostatky v softwarových aplikacích. Jeho cílem je zajistit, že software odolá škodlivým útokům, neoprávněnému přístupu a chrání integritu dat. Toto testování zahrnuje ověřování souladu softwaru s bezpečnostními standardy, hodnocení bezpečnostních funkcí a mechanismů, a provedení penetračních testů k identifikaci potenciálních slabých míst a zranitelností, které by mohly být využity záškodníky. Jeho konečným cílem je identifikovat bezpečnostní rizika a poskytnout doporučení pro nápravu, s cílem zlepšit celkovou bezpečnost softwarové aplikace. Testeři v rámci tohoto testování simuluji útoky, aby ověřili efektivitu stávajících bezpečnostních opatření a odhalili případné nové zranitelnosti. [15]

Tento scénář poskytuje strukturovaný přístup k bezpečnostnímu testování hry World of Warcraft s cílem ochrany proti útokům hráčů a zajištění integrity a bezpečnosti herního prostředí a dat.

Scénář: World of Warcraft - Ochrana proti útokům hráčů

1. Ochrana proti útokům hráčů na herní servery:
   (a) Testování penetračními testy, jako je SQL injection, na vstupy hráčů při zadávání textových příkazů nebo zpráv do chatu.
   (b) Detekce a blokování neoprávněných pokusů o přístup k herním serverům nebo databázím.
2. Ochrana proti úniku citlivých dat hráčů:
   (a) Šifrovaný přenos dat přes API mezi herním klientem a herním serverem, aby se zabránilo odposlechu nebo úniku citlivých informací, jako jsou hesla nebo osobní údaje.
   (b) Zajištění, že hesla uložená v herní databázi jsou řádně šifrovaná, aby se minimalizovala možnost jejich zneužití v případě úniku dat.

3. Omezení přístupu hráčů k citlivým funkcím a datům:

   (a) Implementace oprávnění a přístupových kontrol, aby uživatelé nemohli mít přístup k citlivým údajům, jako jsou herní účty nebo finanční transakce, na které nemají oprávnění.

   (b) Zablokování možnosti hráčů upravovat nebo získávat neoprávněný přístup k herním souborům nebo nastavením, které by mohly vést k porušení bezpečnosti.

2.4.2 Výkonostní testování

- Stres testování
- Testování stability
- Testování načítání

Testování výkonu je postup, při kterém se hodnotí, jak systém funguje z hlediska odezvy a stability při určitém pracovním zatížení. Tento proces obvykle probíhá za účelem ověření rychlosti, robustnosti, spolehlivosti a velikosti aplikace. Testy výkonu zahrnují sledování několika klíčových ukazatelů, jako jsou doba odezvy prohlížeče, stránek a sítě, doba zpracování požadavků serveru, přijatelné objemy souběžně pracujících uživatelů, spotřeba paměti procesoru, a počet a typ chyb, které se mohou vyskytnout u aplikace. Tímto způsobem testování výkonu shromažďuje a hodnotí všechny aspekty, které ovlivňují chod a efektivitu aplikace. [16]

2.4.3 Testování použitelnosti

- Výzkumné testování
- Testování prohlížeče
- Testování přístupnosti

Jedná se o uživatelský výzkum, který hodnotí zkušenosti uživatele při interakci s webovou stránkou nebo aplikací. Tento proces pomáhá návrhářům a produktovým týmům posoudit, jak intuitivní a snadno použitelné jsou produkty. Proces testování použitelnosti odhaluje problémy s produktem, které by jinak mohly být přehlédnuty. Skuteční uživatelé jsou požádáni, aby provedli řadu úkolů týkajících se použitelnosti, a výsledky, míra úspěšnosti a cesty potřebné k dokončení úkolů jsou analyzovány. Tímto způsobem se identifikují potenciální problémy a oblasti, které je třeba zlepšit. Konečným cílem testování použitelnosti je vytvořit produkt, který efektivně řeší potřeby uživatelů a

umožňuje jim dosáhnout svých cílů s pozitivní uživatelskou zkušeností. Pro správné testování je potřeba mít následující kroky: [17]

1. *Příprava a stanovení cílů:* Před testem je důležité jasně stanovit cíle testování a připravit výzkumný plán, který se zaměřuje na klíčové aspekty hry, jako jsou ovládání, UI, obtížnost a zábava.

2. *Výběr vhodných hráčů:* Při pozvání hráčů je důležité zohlednit jejich zkušenosti s daným žánrem a platformou, aby byly výsledky co nejrelevantnější.

3. *Průběh testování:* Testování by mělo probíhat v neutrálním prostředí s moderátorem, který bude věnovat pozornost chování hráčů a jejich reakcím na hru.

4. *Analýza a zpracování výsledků:* Po skončení testů je důležité pečlivě analyzovat získané data a identifikovat klíčové oblasti pro zlepšení hry.

2.5 Zbylé oblasti

- API testování
- Testování zpětné kompatibility
- Regresní testování

Kromě Funkcionálního a Nefunkcionálního testování jsou i další okruhy testování, které je potřeba v některých případech otestovat. Může do toho spadat Agilní testování, testování konfigurací, instancí, uživatelského prostředí nebo například Testování spolehlivosti.

Příklad API testu uložení klienta přes Postman:

```
var jsonData = JSON.parse(responseBody);
postman.setEnvironmentVariable("clientID", jsonData.PrimaryID);

console.log("CostumerID:",jsonData.PrimaryID);


pm.test("Response status code is 200", function () {
    pm.expect(pm.response.code).to.equal(200);
});


pm.test("PrimaryID is not empty", function () {
    const responseData = pm.response.json();

    pm.expect(responseData.PrimaryID).to.exist.and.to.not.be.empty;
});


pm.test("PrimaryID is stored in the environment variable 'clientID'",
    function () {
    const responseData = pm.response.json();

    pm.expect(responseData.PrimaryID).to.exist.and.to.be.a('string');
    pm.environment.set("clientID", responseData.PrimaryID);
});


pm.test("IsValid field is true", function () {
    const responseData = pm.response.json();

    pm.expect(responseData.IsValid).to.be.true;
```

```
});

pm.test("Errors field should be null", function () {
    const responseData = pm.response.json();

    pm.expect(responseData.Errors).to.be.null;
});
pm.test("Warning field should be null", function () {
    const responseData = pm.response.json();

    pm.expect(responseData.Warning).to.be.null;
});
pm.test("Messages field should be null", function () {
    const responseData = pm.response.json();

    pm.expect(responseData.Messages).to.be.null;
});
```

3 HERNÍ PRŮMYSL

Díky hrám jako je Pong, Pac-man nebo Space Invaders se herní průmysl dokázal od 70. let 20. století posunout do moderní doby, kdy je herní průmysl jeden z největších vůbec. Jeho vliv na kulturu, sociální sítě a zábavu nelze podceňovat. Hry nyní poskytují jednu z nejzajímavějších forem zábavy pro více než tři miliardy lidí po celém světě. Technologicky se inovují i platformy, na kterých se samotné hry hrají nebo streamují. Mezi nejpoužívanější platformy patří PC, Playstation nebo Xbox. Na streamování se zase dá využít služby jako Twitch, Youtube nebo Mixer. [19]

3.1 Herní platformy

Počet hráčů se nejspíše pohybuje až ke 3 miliardám. Leč je komunita obrovská, tak jedním z hlavních klíčových aspektů je volba herní platformy. Navzdory rostoucí tendenci k vývoji her, které jsou přístupné na více zařízeních, zůstává část publika různých herních platforem relativně specifická.

3.1.1 PC

Jedná se o nejoblíbenější herní platformu pro vývojáře. Organizátoři se v roce 2021 zeptali 3000 vývojářů a přes 58 % dotázaných vybralo právě PC.[20] Osobní počítače představují zařízení, která slouží k různým účelům, včetně hraní her. S větší flexibilitou a možností přizpůsobení, než herní konzole umožňují osobní počítače upgradování různými komponenty, běh na různých operačních systémech a podporu různých periférií. Oproti konzolím mají osobní počítače také přístup k širšímu výběru her, zejména nezávislým hrám a modifikacím. Je však třeba poznamenat, že počítače mohou být nákladnější, komplikovanější a náchylnější k technickým problémům než herní konzole. [21, 19]

3.1.2 Konzole

Videohry začaly s konzolovými hrami, které jsou považovány za pravděpodobně nejznámější a nejstarší formu. V počátcích ovládaly trh s konzolemi firmy Nintendo a Sega, ale dnes dominují konzolového hraní Xbox, Playstation a Nintendo, kteří poskytují příležitost vyzkoušet si nejnovější herní technologie i jejich vlastní tituly. [21, 19]

Technologický pokrok byl vždy dobře viditelný na konzolovém trhu. V počáteční fázi byly konzole vybaveny jednoduchými joysticky, které byly dostatečně pokročilé pro hraní jednoduchých her jako Pong nebo Pac-man. V současné době jsou konzole více přizpůsobeny hráčům, aby je co nejvíce pohltity do hry. Například v roce 2005 vydal Microsoft konzoli Xbox 360. Jeho velkým tahákem byl Kinect, který umožnil

hráče sledovat a jeho pohyb implementovat do hry. Dalším příkladem může být ovladač Playstationu 5. DualSence byl vyvinut, aby podporoval haptickou odezvu do ovladače nebo přizpůsobivou zpětnou vazbu do zadních páček. Konzole jsou zároveň navrženy tak, aby podporovaly televizní obrazovky s rozlišením 4K. [21, 19]

3.1.3 Cloud

Cloudové hraní představuje metodu, jak hrát videohry prostřednictvím vzdálených serverů umístěných v datových centrech. Přímo do počítače či konzole není nutné stahovat a instalovat žádné hry. Streamovací služby vyžadují stabilní připojení k internetu, což umožňuje odesílání informací o hře do aplikace či prohlížeče nainstalovaného na přijímajícím zařízení. Hra běží a je renderována na vzdáleném serveru, ale hráč s ní interaguje lokálně na svém vlastním zařízení, kde je mu také zobrazena vyrenderovaná scéna. Mezi největší hráče v cloud gamingu je Amazon, Google a Xbox. [22]

3.1.4 VR/AR

VR (Virtual reality) a AR (Augmented reality) jsou jedním z nových odvětví herního průmyslu, kdy využívají buď vlastní nebo reálný svět, aby hráče vtáhly do děje. AR hry v roce 2022 měly podíl na trhu ve výši 8,4 miliardy dolarů. VR v oblasti her získala v roce 2022 tržní podíl 12,13 miliardy dolarů. [23, 21]

Rozšířená realita je technologie umožňující integraci digitálních obrazů a informací do fyzického prostředí. Využívá displeje a kamery, které vytváří vylepšenou verzi reálného světa pomocí zvukových efektů, smyslových podnětů, digitálních vizuálních prvků atp. Mezi jednou z nejznámějších her je Pokemon Go. [23, 21]

Virtuální realita je schopnost vnímat virtuální prostředí jako reálné. Z technického hlediska je virtuální realita ve 3D prostředí, kompletně generované počítačem. Uživatelé mohou toto prostředí prozkoumávat, provádět akce, provádět změny, komunikovat s postavami a manipulovat s objekty. Toho je docíleno pomocí helmy, kterou si hráč dá na hlavu a za pomoci displejů a ovladačů prezentuje tuto realitu přímo do očí hráče. Mezi oblíbené hry patří Beat saber. [23, 21]

3.1.5 Mobilní platformy

Mobilní zařízení zahrnují chytré telefony, tablety a kapesní konzole. Mobilní hry se staly nejdostupnější a nejoblíbenější formou zábavy, protože jsou přístupné pro každého s mobilním zařízením kdykoli a kdekoli. Nicméně mobilní hraní může mít své nevýhody, včetně omezené výdrže baterie, úložného prostoru a výkonu. Předpokládá se, že do roku 2027 bude hrát mobilní hry více než polovina celosvětové populace. Aktuálně mobilní hry tvoří přibližně 45 % celkového tržního podílu herního průmyslu. [21, 19]

3.1.6 Zhodnocení rozdílů platforem z hlediska testování a kontroly kvality:

PC: *Flexibilita a různorodost hardwaru:* Kvůli různorodosti hardwaru mezi jednotlivými počítači může být testování náročné. Je třeba zajistit, že hra funguje správně na různých konfiguracích. *Široká škála periférií:* Testování musí zahrnovat kompatibilitu s různými periferními zařízeními, jako jsou myši, klávesnice, gamepady atd. *Různé operační systémy:* Hra musí být testována a ověřena na různých operačních systémech, jako jsou Windows, macOS a Linux.

Konzole: *Standardizované prostředí:* Konzole mají standardizované hardwarové a softwarové prostředí, což zjednodušuje testování a zajišťuje konzistentní uživatelský zážitek. *Jednotná platforma:* Vývojáři mají k dispozici jednotné vývojové nástroje a prostředí, což usnadňuje testování a optimalizaci her. *Omezená variabilita:* I když jsou konzole standardizované, mohou existovat rozdíly mezi různými verzemi konzolí, které vyžadují testování a optimalizaci.

Cloud: *Stabilní internetové připojení:* Cloudové hraní vyžaduje stabilní internetové připojení pro streamování her. Testování musí zahrnovat odolnost vůči různým úrovním připojení a latence. *Vzdálené servery:* Testování musí zahrnovat spolehlivost a výkon vzdálených serverů, které poskytují hry. *Kompatibilita s různými zařízeními:* Hry musí být testovány a optimalizovány pro různé typy zařízení, jako jsou počítače, konzole, chytré televize atd.

VR/AR: *Optimalizace výkonu:* Testování musí zahrnovat optimalizaci výkonu a grafiky pro zajištění plynulého zážitku ve virtuální realitě. *Testování senzorů a ovladačů:* Hry využívající VR/AR vyžadují testování senzorů a ovladačů pro zajištění přesného a spolehlivého ovládání. *Prevence Motion sickness:* Testování musí zahrnovat preventivní opatření proti Motion sickness a zajištění pohodlného zážitku pro hráče.

Mobilní platformy: *Různá hardwarová specifikace:* Testování musí zahrnovat širokou škálu hardwarových specifikací od různých výrobců a modelů mobilních zařízení. *Omezené prostředí:* Mobilní zařízení mají omezený výkon a bateriovou výdrž, což vyžaduje optimalizaci her a testování na těchto zařízeních. *Různé operační systémy a verze:* Testování musí zahrnovat různé verze operačních systémů (Android, iOS) a jejich aktualizace pro zajištění kompatibility a stabilitu her.

3.2 Vývoj v herním průmyslu

Vývoj softwarových her s sebou nese mnoho výzev a komplikací v rámci procesu softwarového inženýrství. Vedle technologických a funkčních požadavků zahrnuje také rozmanité aktivity v oblasti kreativních uměleckých disciplín, jako jsou storyboarding, design, vylepšování animací, implementace umělé inteligence, tvorba videí, scénáře, zvuková tvorba, marketing a nakonec i aspekty spojené s prodejem. [3, 5]

Proces GDSE (Game Development Software Engineering) se liší od tradičního softwarového inženýrství, a všechny fáze navrženého životního cyklu GDSE mohou být sloučeny do tří hlavních fází: preprodukce, produkce a poprodukce. Fáze preprodukce zahrnuje testování proveditelnosti cílových herních scénářů, včetně inženýrských požadavků na marketingové strategie. Fáze produkce zahrnuje plánování, dokumentaci a implementaci herních scénářů s zvukem a grafikou. Poslední fáze poprodukce zahrnuje marketing, testování a reklamu na hru. Kvůli vysoké konkurenci a extrémní poptávce na trhu někdy herní vývojové společnosti zkracují svůj vývojový proces, aby byly první na trhu. Tato redukce vývojového procesu jednoznačně ovlivňuje kvalitu hry. Kvůli těmto složitým úlohám řízení projektu se herní inženýrský proces odchyluje od tradičního softwarového vývoje. Je tedy nyní důležité zkoumat výzvy nebo problémy, kterým čelí herní vývojové organizace při tvorbě her dobré kvality. [3, 5]

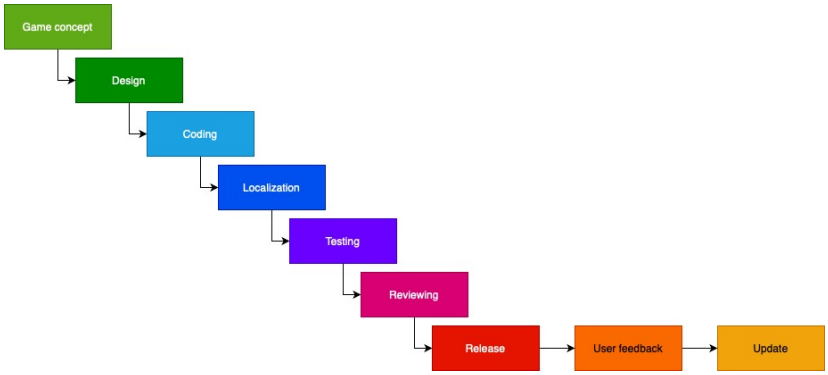
Vývojovou metodu lze definovat jako systematický postup k dosažení cíle produkce funkčního produktu v rámci rozpočtu a harmonogramu. Existuje několik metodologií používaných pro vývoj a design her. [3, 5]

3.2.1 Waterfall

První z nich je metoda Waterfall, která je také běžně používána v tradičním softwarovém vývoji. Na rozdíl od herních projektů jsou aktivity ve fázi preprodukce dokončeny a následně jsou aktivity ve fázi produkce prováděny "vodopádem". Aktivity jsou nejprve odděleny na základě funkcí a aktiv, které jsou poté přiřazeny k příslušným týmům. Tým odpovědný za požadavky věnuje značné množství času definici funkcí a činnostem na předním konci, což naznačuje pozdní implementaci úrovně a mechanismů. Nicméně v metodě Waterfall je obtížné vrátit jakoukoli činnost. Tuto metodu reprezentuje Obrázek 3.1. [3, 5]

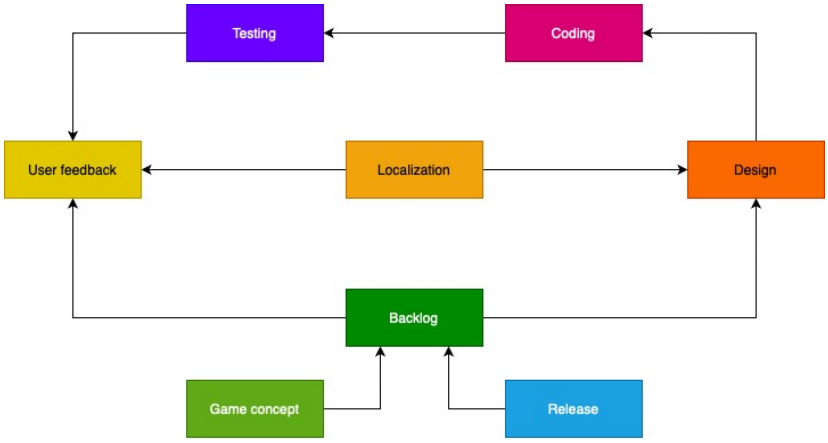
3.2.2 Agile

Druhá metodologie vývoje se více používá v herních studiích a je více běžná při vývoji her. Metoda je silně iterativní a neklade důraz na specifikaci. Fáze produkce je rozdělena na malé iterace, které se zaměřují na nejdůležitější funkce. Celý tým se setká na začátku každé iterace a stanoví si jasné cíle. Na konci každé iterace jsou výsledky komunikovány



Obrázek 3.1 Waterfall metoda s lokalizací

klientům. Tyto metody podporují různé týmové cykly a dynamiku prostřednictvím denních setkání. Nejčastěji používanými agilními metodologiemi ve vývoji her jsou Extreme Programming (XP), Rapid prototyping a Scrum. Tuto metodu reprezentuje Obrázek 3.1. [3, 5]



Obrázek 3.2 Agile metoda s lokalizací

3.2.3 Jednotný proces vývoje

Další tradiční metodou softwarového inženýrství (SE) je Jednotný proces vývoje, který se zaměřuje zejména na analýzu požadavků a jejich transformaci do funkčních softwarových komponent. Dokument analýzy požadavků obsahuje definici koncepce hry, případy užití a definice aktiv. Metoda zahrnuje pět oblastí: požadavky, analýzu, návrh, implementaci a testování. Jednotný proces je postaven na filozofii čtyř klíčových prvků: iterativní a inkrementální, orientovaný na případy užití, orientovaný na architekturu a orientovaný na rizika. [3, 5]

4 HERNÍ TESTOVÁNÍ

Testování v herním průmyslu obsahuje mnoho specifikací a samostatné hry jsou určeny pro širokou paletu zákazníků. Za zmínění stojí hry pro jednoho nebo více hráčů, různé žánry samostatných her nebo rozdílné platformy, na kterých se hry mohou nacházet.

Prudký rozvoj videoherního průmyslu vedl k jejich distribuci na mnoha různých platformách. Hry se staly rozsáhlejšími, složitějšími a náročnějšími na technické prostředky. Současně se výrazně rozšířilo publikum hráčů, kteří se stali stále náročnějšími na kvalitu her. Pro videohry a vývoj her platí běžná rizika softwarových projektů a produktů. V důsledku toho hry, stejně jako ostatní software, podléhají testování. Některá rizika jsou specifická pro oblast herního softwaru a měla by jim být věnována zvýšená pozornost. Mezi rizika můžeme zařadit: multiplatformní problémy, složitost předvídání účinnosti herních mechanik nebo závislost na subjektivních názorech hráčů a tím i úspěchem na trhu. [2]

Na druhou stranu testeři mohou objevit nedostatky například v architektuře herního softwaru na počátku životního cyklu vývoje, ve zvukovém návrhu nebo v textu videohry, která je již připravena k vydání. [2]

Závady spojené s testováním videoher vznikají v různých oblastech, specifických pro tuto kategorii produktů. Mezi tyto problémy patří například problémy s grafikou a animací, porušení fyzikálních zákonů ve virtuálním světě videoher, chyby v umělé inteligenci, nepřesnosti ve vyprávění děje nebo nesprávné chování prvků rozhraní. Identifikace těchto vad je sice často snadná, avšak přesné reprodukce kroků vedoucích k nim může být obtížná. Tato složitost činí testování her náročným úkolem. Kvalitní tester by měl navrhnout testy, které reflektují zamýšlené herní scénáře, ověřit, zda splňují specifikaci, analyzovat možné odchylky a podrobně zaznamenat výsledky. [2]

Při testování her, stejně jako u jakéhokoli softwaru, je důležité identifikovat zjevné i implicitní chyby, které se mohou objevit v důsledku neobvyklých akcí prováděných uživatelem. Například, pokud uživatel namísto boje s protivníky a hledání klíče k zamčeným dveřím shromažďuje krabice a poskládá je tak, aby se vyhnul souboji. Takové chování není v souladu s původním záměrem hry. Další chyby mohou vzniknout při nesprávném nastavení interakcí s objekty, jako je např. 3D model budovy ve hře. Ten při špatném nastavení může hráči umožnit projít zdí a získat výhody. To zdůrazňuje důležitost provádění negativního testování u herních projektů, zejména s ohledem na hráče, kteří hledají chyby a využívají je k vlastnímu prospěchu. [2]

Herní průmysl je víc než jakýkoli jiný, závislý na subjektivním a často emocionálním hodnocení koncových uživatelů. Zatímco pro software navržený k řešení podnikových problémů je klíčovým faktorem jeho funkčnost, pro herní software je nejdůležitějším

faktorem úroveň zájmu uživatele a jeho dojmy z herní seance. Hráči mohou přehlédnout některé funkční chyby, ale pokud se hra ukáže být nudná, monotónní nebo používá zastaralou grafiku, uživatelé ji přestanou hrát. Značná část publika videoher rozhoduje o koupi nebo používání produktu především na základě zpětné vazby od herních kritiků, recenzentů a jiných uživatelů. [2]

4.1 Životní cyklus herního vývoje

Herní vývoj není bohužel podle Pressmana [18] klasickou reprezentací softwaru, takže ani životní cyklus softwarového vývoje (SDLC) nesedí na jeho klasickou reprezentaci. Zatímco SDLC je systematickým procesem inženýrství pro vývoj softwaru, hra není čistým produktem čistého inženýrství. Hra není pouze čisté umění, tvoření kreativity a imaginativního myšlení, ale hra je spíše jako řemeslo, vytvořené kombinací prolínajících se multidisciplinárních aspektů, od umění, hudby, programování, herectví a správy a integrace těchto aspektů. Proto vývoj hry vyžaduje specifické směrnice, které řídí její vývojový proces, tzv. herní vývojový cyklus (GDLC). Existuje mnoho postupů, jak se GDLC uplatňuje v projektu, publikovaných jak nezávislými (indie) herními studii, tak známými společnostmi. Neexistuje GameDLC, který by mohl být brán jako "ten pravý". Vždy je potřeba jej přizpůsobit projektu. [5, 3]

4.1.1 Životní cyklus softwarového vývoje

Životní cyklus vývoje softwaru (SDLC), známý také jako modely softwarového procesu, je strategie vývoje, která zahrnuje procesy, metody a nástroje, které se používají k vytvoření softwaru. [5, 3]

Typický životní cyklus vývoje softwaru. Analýza souvisí se shromažďováním a měřením uživatelských požadavků na vytvoření softwarových specifikací. V návrhové fázi jsou tyto požadavky převedeny do podrobnějších modelů a reprezentací softwarových modulů. Během generování kódu nebo implementace jsou modely převedeny do zdrojového kódu a spustitelné aplikace. Nakonec se provádí testování, aby se zajistilo, že všechny prvky správně fungují a splňují specifikaci. [5, 3]

4.1.2 Blitz Games Studios GDLC

Společnost Blitz Game Studios má založený svůj herní životní cyklus na Obrázku 4.1. Cyklus představuje šest fází, které jsou v lineárním směru. Studio stálo za hrami jako WarGames, Firo And Klawd nebo Bratz: The Movie. [5, 25]

Nejprve se začne přípravou (Pitch). Vytvoří se herní design, který je buď originální, nebo vázán licencí (např. při tvorbě hry na motivy filmu). v první fázi je pouze pár

Obrázek 4.1 Blitz game development life cycle

seniorních designérů a výtvarníků. Zároveň se vyjednává s vydavatelem o časovém plánu, rozpočtu nebo specifikách. [5, 25]

Následuje fáze preprodukční (Pre-Production). V této fázi se podrobně naplánuje herní design, herní engine (vývoj vlastního/použít existující řešení), nástroje pro umělce, designery a animátory. V případě nových prvků (např. nový žánr nebo platforma) se vytváří prototyp, aby vývojáři měli šablonu, podle které budou programovat. [25]

V Produkční (Production) fázi se každých čtyři až šest týdnů pošle aktuální verze hry dodavatelům. Jsou to milníky, které jsou ve smlouvě jasně definované, co mají obsahovat. V této fázi se také vytváří dokumentace, která bude každým milníkem přidávat více a více komplexity hry samotné. Důležitá je též organizace. V této fázi nastoupí na projekt mnoho nových lidí, kteří skládají dohromady různé assety jako textury, animace, hudba, osvětlení atp. [5, 25]

Fáze Alfa je charakteristická tím, že hra již bývá v hratelném stavu. Stále se v ní nachází zástupné prvky jako nedokončené textury nebo chybějící zvuk, ale hru lze odehrát od začátku do konce. [5, 25]

Beta verze je již kompletně hotová hra, která se už pouze ladí a opravují se v ní bugy. Během této fáze se pomalu začne tým zmenšovat a např. animátoři, výtvarníci a designeři přejdou na nové projekty. Zůstávají převážně vývojáři a testeři, kteří musí testovat a opravovat chyby. [5, 25]

V konečné fázi se hra dostane do Masteru, kdy se hra rozmnoží a distribuuje do obchodů. [5, 25]

4.1.3 Arnold Hendrick's GDLC

Na Obrázku 4.2 lze vidět GDLC Arnolda Hendricksna. Cyklus je stejně jako u Blitz Game studios lineární. Rozdíl ovšem je, že obsahuje pouze pět fází. [5, 26]

Obrázek 4.2 Arnold Hendrick's GDLC

Ve fázi prototypu (Prototype) se vybírá a testuje herní engine, softwarové nástroje jako IDE a nástroje pro build projektu. Vytvoří se výtvarný styl a design koncept, první návrh designové dokumentace a počáteční návrh designu. Dále se navrhne herní

prototypy, počáteční koncepční výtvarné návrhy a 3D výtvarné/animační prototypy. [5, 26]

V průběhu preprodukční (Pre-Production) fáze je nutné vytvořit dvě až tři plně hratelné zóny (úrovně) včetně odměn a postupů. Jednu zónu postavit stejnými metodami, jaké se používají při výrobě – jen tak lze přesně odhadnout výrobní náklady. Klíčové koncepty je třeba dokončit. Návrhová dokumentace a výrobní specifikace (pro programování, design, výtvarné zpracování, zvuk a QA) musí být rovněž dokončeny. [5, 26]

V produkční (Production) fázi se vytvoří zbývající zóny a hra se dodělá na takovou úroveň, aby byla kompletní a obsahovala veškeré zvuky, úkoly, umělecké prostředky, scripty pro umělou inteligenci atp. Často se v této fázi přibírají noví lidé do týmu (outsourcing nebo převedení z jiného týmu), aby pracovali na různých sekcích podle stanoveného plánu. Také se dopíše specifikace a vytvoří se procesy pro CS nástroje, účtování a aktualizační systémy. [5, 26]

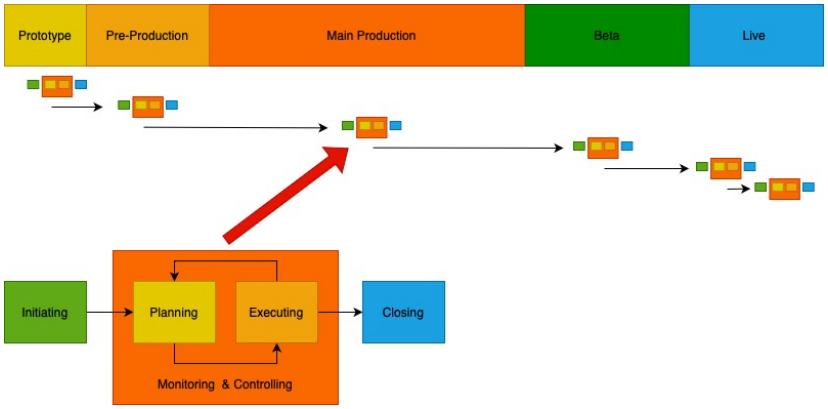
Beta je charakteristická stabilizací, opravou chyb a laděním. Ladí se hratelnost a řeší se různé provozní chyby, které mohou nastat. Dokončí se systémy pro fakturaci, CS a aktualizaci. Beta končí dnem, kdy se začne promovat tzv. "otevřená beta". [5, 26]

V poslední fázi (Live) je hra vystavena a v nepřetržitém provozu. Začíná "otevřenou betou"a tu podporují dva týmy. První se stará o opravy chyb a každodenní provozní záležitosti. Druhý zase o měsíční aktualizace a dlouhodobá rozšíření. [5, 26]

Podle Hendricksna je klíčové mít předchozí fázi životního cyklu vždy uzavřenou, protože v opačném případě nastane chaos a projekt dopadne katastrofou. Toho by se měli držet veškeré větší hry (např. RPG, MMO atd.) protože práce na jednotlivých fázích je rozdílná oproti klasickému projektovému životnímu cyklu. [5, 26]

Například Scrum může být vhodný pro fáze předpřípravy a prototypování. Zde díky zpracování různorodých řešení potřeb a cílů lze efektivně využít prioritizaci cílů. Ovšem produkční fáze je odlišná. V ní se musí tým vypořádat s ohromným množstvím drobných položek (AI, umělecké assety, modely, zvuk, hudba, uživatelské rozhraní atd.) které musí vyrobit a implementovat do hry samotné. Každá tato položka má svůj vlastní miniaturní proces vytvoření, schválení a otestování. To se u velkých her za pomocí různých sledovacích nástrojů, tabulek nebo databází, kde se vše zaznamenává a sleduje. [5, 26]

Východiskem tedy může být to, že každá fáze herního životního cyklu je samostatný projekt (vývoj hry je tedy multifázový projekt), kdy první musí skončit předešlý krok, aby mohl být spuštěn další. Tím se i lépe sleduje postup vývoje a zlepšuje to vztahy mezi vydavatelem a vývojáři. Na Obrázku 4.3 lze vidět klasický projektový životní cyklus a jeho implementace do toho herního. Diagram ukazuje, že inicializace a plá-

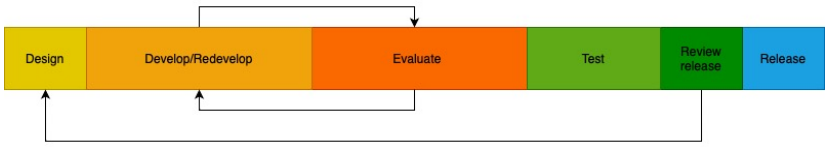


Obrázek 4.3 GDLC s projektovým životním cyklem

nování další fáze začíná v pozdější části aktuální fáze. Například během preprodukční fáze může výrobce potřebovat najít externího dodavatele grafických prací, který by pomohl zvládnout masu výtvarných prostředků potřebných během produkce. Režim "Live"obsahuje několik opakování projektu. To představuje sérii aktualizací/rozšíření hry, z nichž každá je považována za vlastní projekt. Přesto se dá říct, že metody projektového řízení mohou koexistovat s téměř jakýmkoliv vývojovým procesem jako například Scrum, Vodopád, Iterace, atp. [5, 26]

4.1.4 Doppler Interactive GDLC

Metodika formalizuje potřebu experimentálního vývoje. Cyklus je agilního typu a doporučuje se pro všechny menší týmy, které chtějí mít robustnější systém práce. Na prvním místě je potřeba experimentovat, selhávat a poučit se. Teprve poté je potřeba dělat návrh, který je též dost důležitým pilířem celého systému. Na Obrázku 4.4 lze vidět schéma tohoto životního cyklu. [5, 27]



Obrázek 4.4 Doppler Interactive GDLC

V první fázi probíhá návrh (Design). Zde se navrhne herní engine a vymyslí základní koncepce herního světa. Doba trvání může být různá, ale v dokumentu se specificky mluví o jednom týdnu na jednu fázi. [5, 27]

Následující fáze je o vytvoření (Development) prototypu enginu. Stačí, ať splňuje základní požadavky, ale musí být funkční a prokázat svůj koncept. Po vytvoření enginu se začne pracovat na tom, aby byl použit ve vývoji herních funkcí. Například se jedná o funkce typu komunikace postav, pohybu vozidel nebo aplikace fyziky na předměty. [5, 27]

Ve třetí fázi (Evaluate) se zhodnotí samotný engine i jeho aplikování do hry a vyhodnotí se stav. Pokládají se otázky typu, Co bylo zbytečné? Co trvalo dlouho? Které funkce jsou nezbytné? Nebo které funkce byly obtížné na implementaci? Po kompletním zhodnocení se projekt přesune do předešlé fáze a začne znova vyvíjet (přeprogramovávat) engine nebo funkcionality. Vše se zaznamenává do dokumentace. [5, 27]

Po nějaké době vznikne demo, které se předá na interní testování (Test). V této fázi se převážně ladí a opravují bugy, na které interní tým testeru narazí. Následně se projekt dostane do fáze, kdy se uvolní jako veřejná beta pro hráče (Review elease). Tady vývojáři sbírají postřehy, bugy a recenze na hru samotnou a vrací se do první fáze návrhu, kdy podle zpětné vazby hru více přizpůsobí samotným hráčům. V konečné fázi (Release) hra vyjde ve verzi 1.0 a následuje oprava chyb a příprava budoucího obsahu. [5, 27]

4.2 Typy defektů

Společností IBM byl vyvinut systém The orthogonal defect classification (ODC) pro klasifikaci chyb dle kategorií. Systém definuje různé kategorie klasifikací v závislosti na probíhající vývojové činnosti. Kapitola se bude přesněji zabývat osmi typy defektů, které se týkají her. Každá chyba je důsledkem buď nesprávné implementace nebo se funckionalita v kódu nenachází vůbec. [4, 28]

Příklady z této kapitoly jsou ze hry Dark Age of Camelot (DAOC) Verze 1.70i. Dark Age of Camelot je Massively Multiplayer Online Role-Playing Game (MMORPG) hra, která je zaměřena na fantasy s kombinací norské mytologie, irské keltské legendy a tématiku krále Artuše. Hra byla tedy ve vývoji kde se řešily dvě roviny. Oprava chyb a přidávání/úprava nových funkcí. V Patchi byla zaznamenána tato oprava:

"Schopnost Vanish realm nyní hlásí, kolik sekund super-stealth máte k dispozici"

Chyba teda byla hlášena nejspíš stylem:

"Schopnost Vanish realm NEhlásí, kolik sekund máte super-stealth, když je použita"

[4, 28]

Popis schopnosti Vanish pro lepší pochopení: Poskytuje hráči možnost "super stealth"(zmizení), který nelze prolomit. Odstraní DoTs a Bleeds a poskytne imunitu vůči kontrole davu. Hráč získá rychlejší pohyb, ale nemůže po dobu 30 sekund útočit po aplikování schopnosti.

Efekty:

- L1 - normální rychlost, imunita 1s
- L2 - rychlost 1, imunita 2s

- L3 - rychlost 5, imunita 5s
- Typ: aktivní
- Znovu použití: 10 min
- Level 1 - 5
- Level 2 - 10
- Level 3 - 15
- Třídy pro Vanishera: Infiltrator, Nightshade, Shadowblade

4.2.1 Funkce

Chyba funkce ovlivňuje schopnost hry nebo to, jak hráč hru používá. V kódu nastala chyba, kdy buď daná funkce úplně chybí nebo je nesprávně implementována. Zde je teoretický kus kódu, který představuje možnost použít a nastavit schopnost Vanish. [4, 28]

```
1     void HandleVanish(level) {
2         if (level == 0)
3         return; // player does not have this ability so leave
4
5
6         PurgeEffects(damageOverTime);
7         IncreaseSpeed(g_vanishSpeed[level]);
8         SetAttack(SUSPEND, 30SECONDS);
9         StartTimer(g_vanishTime[level]);
10        return;
11        } // oops! Did not report seconds remaining to User
12
13
14
15        ShowDuration(FALSE, g_vanishTime[level]); // Wrong calling
```

Kód obsahuje funkci `HandleVanish`, která se volá v případě použití schopnosti. Pole `g_vanishSpeed` a `g_vanishTime` uchovávají hodnoty pro každou ze tří úrovní této schopnosti a navíc hodnotu 0 pro úroveň 0. Jelikož výsledek platí pro všechny postavy, pole jsou globální. Hodnoty velkými písmeny jsou konstanty. [4, 28]

V tomto případě je chybějící volání funkce příkladem kategorie Funkce. Důvody mohou být různé. Buď se stala chyba při komunikaci a programátor špatně pochopil zadání, nebo tento kód byl zkopírován z jiné schopnosti a následně byl jen trochu upraven. Případně mohla být funkce volána špatně a obsahovala například více parametrů, než měla. [4, 28]

4.2.2 Přiřazení

V případě Přiřazení se jedná o chybu, kdy může být výsledkem buď nesprávné inicializace/nastavení hodnoty. Nebo pokud chybí přiřazení dané hodnoty. Mnoho přiřazení nastává na začátku hry, úrovně nebo herního režimu. Zde je pár příkladů, které mohou v případě špatné inicializace vnést výhodu ve prospěch hráče nebo CPU: [4, 28]

- RPG:
  - Výchozí místo na mapě.
  - Inicializace dat o dovednostech, předmětech.
- Strategie:
  - Cíl pro aktuální scénář.
  - Prvotní přidělení jednotek, surovin.
- Sportovní hry:
  - Rozpis a sestava týmu.
  - Inicializace scóre.

```
1    ABILITY_STRUCT realmAbility;
2
3        realmAbility.ability = VANISH_ABILITY;
4        reamAbility.purge = DAMAGE_OVER_TIME_PURGE;
5        realmAbility.level = g_currentCharacterLevel[VANISH_ABILITY];
6        reamAbility.speed = g_vanishSpeed[realmAbility.level];
7        realmAbility.attackDelay = 30SECONDS;
8        realmAbility.duration = g_vanishTime[realmAbility.level];
9        realmAbility.displayDuration = FALSE; // wrong flag value
10
11   HandleAbility(realmAbility);
```

Pro účely chyby ve schopnosti Vanish je vytvořen tento imaginární kód. Schopnost je aktivována nastavením datové struktury a jejím volám ve funkci pro obecné zpracování schopnosti. Zde se chyba nachází ve špatné inicializaci `realmAbility.displayDuration`, která dostává boolean atribut, který by měl být ve skutečnosti integer nebo float. Případně může inicializace chybět úplně. Stejně jako v případě kategorie Funkce mohla být způsobena špatným pochopením u vývojáře nebo vložením zkopírovaného kódu odjinud, který se špatně upravil. [4, 28]

4.2.3 Kontrola

Kategorie Kontroly charakterizuje chyby způsobené neověřením dat. Může zde patřit jak chybějící podmínka nebo její špatná definice. V jazyce C to může být například: [4]

- špatné porovnávání hodnot - místo "=="se použije pouze "=",
- Hodnota (*pointer) porovnávaná s NULL místo adresy (pointer) - buď přímo z uložené proměnné, nebo jako vrácená hodnota z volání funkce,
- Porovnávání "off by one"- místo "<="se použije "<"

```
HandleAbility (ABILITY_STRUCT ability) {

    PurgeEffect(ability.purge);
    if (ability.attackDelay > 0)
        StartAttackDelayTimer(ability.attackDelay);

    if (ability.immunityDuration == TRUE)
    // should be checking ability.displayImmunityDuration!
        DisplayAbilityDuration(ability.immunityDuration);
}
```

Následující scénář reprezentuje imaginární kód, kde podmínka má správně kontrolovat `ability.displayImmunityDuration` a ne `ability.immunityDuration`. Chyba mohla vzniknout stejně jako v předešlých dvou kategoriích. [4, 28]

4.2.4 Načasování

Některé procesy mohou být závislé na času (například ukládání informací na disk počítače). V případě Načasování bývají problémy tedy se sdílením zdrojů v reálném čase. Uživatelsky přívětivý proces během těchto stavů může být vložením "cut scény", úvodní obrazovka při zapnutí hry nebo animace točícího kolečka, která říká, že se hra právě ukládá. Tímto způsoben není nijak bráněno závislým procesům, které je potřeba pro zapsání/nahrání dat. [4, 28]

Načítání zvuku a grafiky je také závislé na správném načasování. Mnoho z těchto funkcí je nyní zapracováváno v herním hardwaru, ale SW může stále potřebovat čekat na nějaký druh oznámení. Například příznak, který se nastaví na událost, která se odešle obsluze události nebo funkci, která se zavolá, jakmile jsou data připravena k použití. [4, 28]

```
    //Defective code:
    StartAnimation(VANISH_ABILITY);
    ShowDuration(TRUE, g_vanishImmunityTime[level]);

    //Correct code:
    StartAnimation(VANISH_ABILITY);
    while(g_animationDone == FALSE); // wait for TRUE
        ShowDuration(TRUE, g_vanishImmunityTime[level]);
```

V kódu výše je scénář poruchy časování. Funkce `StartAnimation(VANISH_ABILITY)` spustí animaci pro dosazení schopnosti Vanish a globální proměnná `g_animationDone` je nastavena tehdy, až skončí animace. Pokud je `g_animationDone` TRUE, zobrazí se doba trvání. K chybě načasování dojde tehdy, pokud funkce `ShowDuration` je volána, aniž by se čekalo na potvrzení, že animace byla dokončena. [4, 28]

4.2.5 Build chyby

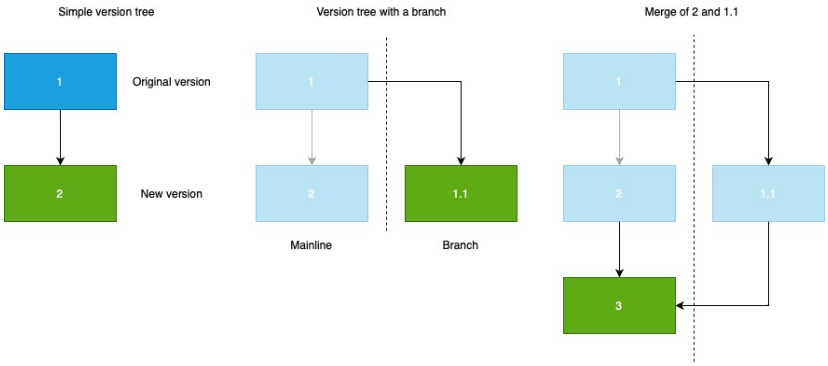
Tyto chyby vznikají v důsledku chyb při používání systému knihovny zdrojového kódu hry, při správě změn v souborech hry nebo při určování a kontrole toho, které verze se sestavují. Při sestavě (Buildu) se kompiluje zdrojový kód s herními prostředky (grafika, zvuk, text, ...) za účelem vytvoření spustitelného souboru hry. Používá se na to

software pro správu verzí (Bitbucket, GitHub, Unity Version Control), který spravuje a kontroluje soubory hry a pro každou verzi vytváří specifický identifikátor. [4, 28]

V confic spec (specifikace konfigurace) je pak seznam verzí jednotlivých souborů, které se mají sestavit. Jelikož je tento proces náchylný k chybám a je časově náročný, jednotlivé verze se označkují. Podle těchto štítků se pak identifikují skupiny konkrétních verzí. Příkladem štítků může být: [4, 28]

- DevBuild - Verze, která se využívá při opravě chyb nebo implementaci a vyzkoušení nových funkcionalit,
- PcOnly - V případě hry, která je vyvíjena na více platformách,
- TestRelease - Verze, která je pro testery a programátoři si jsou jistí, že funkcionalita je připravena na interní test.

Každý soubor má svou vlastní evoluční cestu zvanou Mainline. Nové verze tohoto souborů, které jsou odvozeny z původní verze se nazývají Branche (větve). Změny mezi více branchemi lze kombinovat s dalšími změnami paralelně díky procesu zvanému Merge (sloučení). Sloučení lze provést manuálně, automaticky nebo za pomoci systémového správce, který může zvýraznit změny mezi dvěma verzemi. Na Obrázku 4.5 lze vidět grafický strom verzí, který reprezentuje tyto náležitosti. V případě použití systému správy konfigurace, nová verze souboru se dá na kontrolu. Po ní se schválí změna a vystaví se nová verze. V opačném případě lze kontrolu zamítnout a ponechat původní verzi beze změny. [4, 28]



Obrázek 4.5 Tři diagramy charakterující různé variace Stromu verzí

Zadání nesprávné verze nebo označení ve specifikaci konfigurace může vést k vygenerování spustitelného souboru hry, ale nebude fungovat tak, jak bylo zamýšleno. Pouze jeden soubor může být špatný, takže chyba se může projevit pouze ve specifické situaci. Další možností je, že specifikace konfigurace je správná, ale jeden nebo

více programátorů neoznačilo správně verzi, kterou bylo třeba sestavit. Je možné štítek vynechat, nechat jej ve dřívější verzi nebo ho zadat špatně. A Někdy je problém ve sloučení. Složitost sloučení se zvýší, pokud jedna verze souboru odstranila část kódu, která byla aktualizována verzí, s níž je slučována. Někdy může i kód naznačit, že je něco špatně (Např. komentáři typu //TODO nebo //WARNING). [4, 28]

Vznik chyby pro nezobrazení trvání Vanish schopnosti můžeme teoreticky přirovnat k pravé části Obrázku 4.5. Stanoví se, že ve verzi 2 funkcionalita nebyla a ve verzi 1.1 ano. Mohla nastat jedna ze tří situací: [4]

- Při sloučení a vytvoření verze 3 se odstranila část kódu, která obsahovala zobrazení doby trvání.
  - Verze 3 se vystavila, ale nezobrazila se nová funkcionalita.
- Při sloučení vznikla verze 3, která obsahovala kód se zobrazením doby trvání
  - Avšak označení specifikace nebylo změněno z verze 2 na verzi 3, takže při sestavě (buildu) byla vystavena verze bez zobrazení.
- Při sloučení vznikla verze 3, která obsahovala kód se zobrazením doby trvání a označení specifikace bylo změněno na verzi 3.
  - Specifikace sestavení byla v kódu fixně nastavena na verzi 2 místo použití označení, takže se nezobrazila doba trvání.

4.2.6 Algoritmus

Chyby mohou nastat i v algoritmech. Algoritmem se rozumí nějaký výpočet nebo rozhodovací proces, který má přinést nějaký výsledek (Např. otevření dveří, zahájení dialogu nebo vrácení čísel). Zde je seznam různých algoritmů podle žánru hry: [4, 28]

- RPG:
  - Dialogové reakce na postavy.
  - Výpočty bonusů, zkušenostních bodů.
- Strategie:
  - Umožnění používání nových jednotek, zbraní, technologií, zařízení atd.
  - Výpočty poškození a účinku.
- Logické hry:
  - Určení podmínek pro konec kola a postupu do další úrovně.

- Povolení speciálních vylepšení, odměn.

Některé hry ale obsahují více žánrů a různorodých algoritmů. Dalšími oblastmi, mohou být grafické vykreslovací enginy a rutiny, mesh overlay code, z-buffer ordering, detekce kolizí a pokusy o minimalizaci kroků zpracování při vykreslování nových obrazovek. [4, 28]

Pro příklad je situace defektu algoritmu, kdy se hodnota délky trvání schopnosti počítá namísto přebírání z pole. Doba trvání se též nezobrazí, pokud je hodnota 0 nebo nižší. Tudíž v případě výpočtu nulového nebo záporného výsledku doba trvání vůbec neukáže. Trvání imunity je vyjádřeno vztahem: 1 level = 1 sekunda, 2 level = 2 sekundy a 3 level = 5 sekund. Viz vzorec: [4]

```
vanishDuration = (2 « level) - level;
```

Podle specifikace chceme tedy výsledky 1 level to bude 1 level: 2 - 1 = 1, 2 level: 4 - 2 = 2 a 3 level: 8 - 3 = 5. Pokud by ale programátor omylem vložil jiný operátor (např. modulo %) výsledky by byly jiné... 0 - 1 = -1 pro Level 1, 0 - 2 = -2 pro Level 2 a 2 - 5 = -3 pro Level 3. [4]

4.2.7 Dokumentace

Jako vady dokumentace se vyskytují nejčastěji ve fixních souborech hry. Chyba ale není způsobena nesprávným kódem, nýbrž v samostatném zápisu souboru, který je poté volán a využíván programem. Patří zde obsah zvuková, grafický i textový (Dialogy, zvukové efekty, cut scény, postavy, objekty, ...) Můžou mezi ně patřit i řetězové textace, které se například využívají při lokalizacích (různé jazyky hry) nebo zobrazení textu typu "Kolo úspěšně dokončeno"nebo "Wasted". [4, 28]

Následující příklady se nebudou řídit dobou trvání Vanish, ale jinými opravami v Patchi 1.70i. [4]

- Byly provedeny opravy gramatických chyb v popisu schopností.
  - Neuvádí se žádná konkrétní podmínka, za které jsou tyto údaje nesprávné.
  - Chyba je gramatická, takže text byl poskytnut a zobrazen, ale samotný text byl chybný.
- Sabotage ML se již nesprávně neodkazuje na obléhací vybavení.
  - Tento popis se vztahuje na provedení příkazu `/delve` ve hře pro schopnost Sabotage Master Level.
  - Chyba mohla být odstraněna opravou textu nebo delve byl získán pro nějakou jinou schopnost podobnou Sabotáži kvůli chybnému indexu pole ukazatelů (př. vada přiřazení nebo funkce).

4.2.8 Interface

Poslední chybou je chyba Interface. Nastává tehdy, když je problém s přenosem nebo výměnou informací. V kódu hry se chyby rozhraní objevují, když je něco špatně ve způsobu, jakým jeden modul volá druhý. Chyby mají vlastní kategorizaci [4, 28]:

1. Volání funkce s nesprávnou hodnotou jednoho nebo více argumentů,
2. Volání funkce s argumenty předanými v nesprávném pořadí,
3. Volání funkce s chybějícím argumentem,
4. Volání funkce s negovanou hodnotou parametru,
5. Volání funkce s bitově invertovanou hodnotou parametru,
6. Volání funkce s argumentem inkrementovaným oproti zamýšlené hodnotě,
7. Volání funkce s argumentem zmenšeným oproti zamýšlené hodnotě.

Pro příklad se použije metoda z dřívějška `ShowDuration` s následující podobou:

```
void ShowDuration(BOOLEAN_T bShow, int duration);
```

Ta je typu void, tudíž nic nevrací. Získává boolean hodnotu (`bShow`), která určí, jestli se má zobrazit doba trvání a číselnou hodnotu integer (`duration`), která říká, jak dlouho se má zobrazit, pokud je vyšší než 1. [4, 28]

```
1    //Calling a function with the wrong value of one or more arguments
2    ShowDuration(TRUE, g_vanishSpeed[level]);
```

Pro získání doby trvání použito nesprávné globální pole (rychlost místo doby trvání). To může vést k zobrazení nesprávné hodnoty nebo k žádnému zobrazení (pokud je předána hodnota 0). [4, 28]

```
1    //Calling a function with arguments passed in the wrong order
2    ShowDuration(g_vanishDuration[level], TRUE);
```

Datový typ `BOOLEAN_T` je ddefinován jako int. Oba parametry se prohodí, takže první hodnota bude porovnána s TRUE, a druhý parametr TRUE bude porovnáván s číslem. Pokud hodnota trvání neodpovídá definici, pak se nezobrazí žádná hodnota. [4, 28]

```
1    //Calling a function with a missing argument
2    ShowDuration(TRUE);
```

Není zobrazena žádná hodnota trvání. Pokud je výchozí hodnota 0 v důsledku deklarace lokální proměnné v rámci příkazu `ShowDuration`, pak se nezobrazí žádná hodnota. [4, 28]

```
1     //Calling a function with a negated parameter value
2     ShowDuration(TRUE, g_vanishDuration[level] | 0x8000);
```

Je nastaveno, že bit vyššího řádu v hodnotě trvání funguje jako příznak, který musí být nastaven (vznik např. špatnou implementací z dřívějška nebo pozůstatek starší metody). Místo zamýšleného výsledku se ale změní znaménkový bit hodnoty trvání a dojde k její negaci a tím pádem nebude schopnost zobrazena. [4, 28]

```
1     //Calling a function with a bitwise inverted parameter value
2     ShowDuration(TRUE, g_vanishDuration[level] ^ TRUE);
```

Zde vede operace Exclusive OR prováděná na hodnotě trvání. Může se jednat o možný pokus využít některý konkrétní bit v hodnotě trvání jako indikátor pro, zda tuto hodnotu. V případě, že TRUE je rovno 0xFFFF, dojde k invertování všech bitů v hodnotě trvání. To způsobí, že bude předána jako záporné číslo, čímž se změní její hodnota a zabrání se jejímu zobrazení. [4, 28]

```
1     //Calling a function with an argument incremented from its
2    intended value
3     ShowDuration(FALSE, g_vanishDuration[level+1]);
```

Může vést k problému, kdy hodnota úrovně je inkrementována tak, aby začínala prvním prvkem pole. Pokud je `level` 3, může to vést k trvání 0, protože `g_vanishDuration[4]` není definováno. [4, 28]

```
1     //Calling a function with an argument decremented from its
2    intended value
3     ShowDuration(FALSE, g_vanishDuration[level-1]);
```

Zde stejný příklad jako u zvýšení zamýšlené hodnoty, akorát to je naopak. Pokud je `level` 1, mohlo by to vrátit hodnotu 0 a to zabrání zobrazení hodnoty. [4, 28]

4.3 Testovací techniky

Následující sekce je zaměřena na pět testovacích technik, které se využívají při testování her. Nabízí různé pohledy na efektivitu a minimalizaci chyb během testu.

4.3.1 Kombinatorické testování

Rozlišuje se převážně úplné a párové kombinatorické testování. Párové kombinované testování je způsob, jak najít chyby a zároveň mít malé testovací sady vzhledem k množství funkcí, které pokrývají. Párová kombinace znamená, že každá hodnota, která se použije pro testování, musí být alespoň jednou zkombinována s každou další hodnotou zbývajících parametrů. [4, 29]

Parametry jsou jednotlivé herní prvky, které se mají otestovat. Může mezi ně patřit nastavení hry, HW konfigurace, možnosti přizpůsobení, Herní události, ... Vytvořené

testy jsou buď homogenní (parametry pro kombinace jsou stejného typu) nebo heterogenní (parametry pro kombinace jsou různého typu). Příkladem homogenního kombinatorického testu je testování herních nastavení. Výběr mezi různými postavami a vybavením pro konkrétní misi je příkladem heterogenního kombinatorického testu. [4, 29]

Každý parametr obsahuje určité volby (hodnoty). Ty mohou být různého druhu a v rámci jednoho parametru jich může být velké množství. Hodnoty jsou různých druhů:

Výchozí Jsou to výchozí hodnoty, které jsou nastavené v otevření dané sekce. Na jednu stranu by měly být zahrnuty do testů. Jedná se snad o nejčastěji používané hodnoty, které budou nastaveny. Na druhou stranu ale je možné v případě ušetření prostředků a času tyto hodnoty právě z tohoto důvodu vynechat. V tomto případě je ale vhodné je zahrnout alespoň do sady jedné testů. [4]

Výčty Ve hře se vyskytuje mnoho voleb možností, které nemají žádný konkrétní číselný nebo postupný vztah. každá z těchto voleb (tým, bojovník, zbraň, účes atd.) bez ohledu na počet by měla být zastoupena v testech. Je snadné najít chyby, ke kterým dochází nezávisle na tom, jaká konkrétní volba je provedena. Ty, které uniknou, se obvykle vyskytují jen u několika málo voleb. [4]

Rozsah Hráč vybírá ze seznamu nebo řady určité číslo, které něco charakterizuje. Nejčastěji tři konkrétní hodnoty způsobují největší chybovost: nula, minimum a maximum. Pokud se 0 nachází v možnostech zadání, měla by být vždy vybrána do testů. Existují různé důvody, co může způsobit. Např.: [4]

- 0 se často používá pro vyjádření zvláštního významu, například pro označení nekonečného časovače nebo toho, že došlo k chybě.
- Cyklus se může předčasně ukončit nebo může vždy něco provést před kontrolou nuly.

V případě použití minimálních hodnot je také zvýšené riziko nalezení chyby. Například použití minimální doby nemusí umožnit dokončení některých započatých efektů, nebo může způsobit, že některé cíle nebudou dosažitelné. [4]

Maximální hodnoty mohou být pro testera dodatečnou časovou zátěží, neboť v některých situacích musí vynaložit vyšší úsilí, aby jich dosáhl. Kromě testování herních prvků je důležité zahrnout také testy maximálního počtu uložených souborů, maximálního počtu hráčů a maximálního úložného prostoru (disk, kazeta apod.). [4]

Hranice Testování hranic značí testování na pomezí možností dané hry. Může se jednat o hranice města, startovní a cílové čáry, traťové oblasti, časovače misí, vzdálenosti projektilu/šípu atp. Ve hře ESPN NFL 2K5 je funkce CRIB, která uděluje hráčům body za úspěchy během jednoho zápasu, v průběhu sezóny a v průběhu kariéry. Pokud hráč má hru kratší než pět nebo osm minut, nedostane některé odměny a tím vznikají další hraniční hodnoty čtvrtletní délky, které jsou předmětem zájmu. [4]

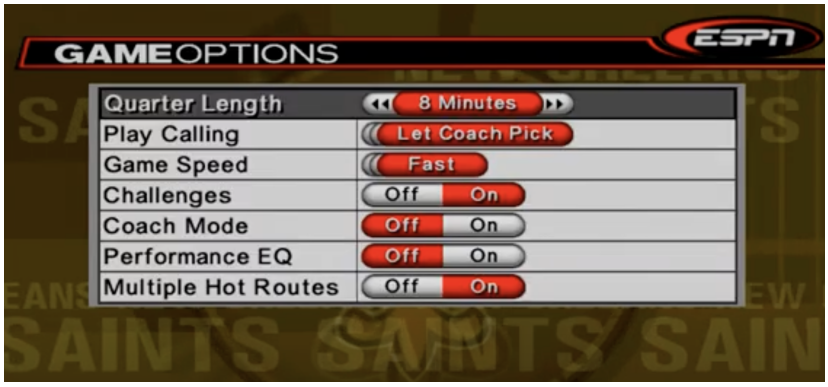
Pro vytvoření párové tabulky je potřeba dodržet tento postup:

1. Vybrat parametr s největším rozměrem.
2. První sloupec se vytvoří tak, že každou hodnotu testu pro první parametr vypíšete N-krát, kde N je dimenze parametru s další nejvyšší dimenzí.
3. Vypíše se další sloupec vypsáním testovacích hodnot pro další parametr.
4. Pro každý zbývající řádek tabulky se uvede do nového sloupce hodnota parametru, která poskytuje největší počet nových dvojic vzhledem ke všem předchozím parametrům zadaným do tabulky. Pokud se taková hodnota nenajde, změní se jedna z dříve zadaných hodnot pro tento sloupec a pokračuje se v tomto kroku.
5. Pokud jsou v tabulce nevyhovující dvojice, vytvoří se nové řádky a vyplní se hodnoty potřebné k vytvoření jedné z požadovaných dvojic. Pokud jsou všechny dvojice splněny, Přejde se zpět ke kroku 3.
6. Vloží se další neuspokojené dvojice pomocí prázdných míst v tabulce, aby se vytvořilo co nejvíce nových dvojic. Přejde se zpět ke kroku 5.
7. Vyplní se prázdná políčka některou z hodnot pro příslušný parametr (sloupec).

Příklad se bude zabývat herním nastavením ve hře ESPN NFL 2K5. Toto nastavení je na Obrázku 4.6. Qarter Length lze nastavit na hodnoty 1 až 15. Defaultně je nastavena hodnota 5. Pro test se použije hodnota 1, 5 a 15. Play Calling má možnosti By Package, By Formation a Let Coach Pick. Game Speed nabízí rychlost Slow, Normal a Fast. Možnost Performance EQ nebude v testu zahrnuta. Nastavení Challenges, Coach Mode a Multiple Hot Routes mají na výběr On a Off. [4]

Vytvoří se tabulka $3^2 2^3$ se třemi parametry a třemi hodnotami: Quarter Length, Play Calling a Game Speed. A se dvěma parametry a třemi hodnotami: Challenges, Coach Mode a Multiple Hot Routes. [4]

Nejprve se začne kroky 1 a 2 a třikrát se vypíšou hodnoty Quarter Length v prvním sloupci tabulky. Jedná se o parametr s největším rozměrem (3). Následně se podle 3 kroku vyplní každá z hodnot Play Calling do prvních tří řádků druhého sloupce. Obsah je znázorněn v Tabulce 4.1. [4]



Obrázek 4.6 ESPN NFL 2K5 Nastavení[44]

Tabulka 4.1 Testovací tabulka I

Quarter Length	Play Calling
1 min	Package
5 min	Formation
15 min	Coach
1 min	
5 min	
15 min	
1 min	
5 min	
15 min	

Dále se pokračuje 4 krokem. Ve čtvrtém řádku druhého sloupce se pokračuje s parametrem, který má největší počet nových párů. Protože se jedná pouze o druhý sloupec, můžete vytvořit pouze jednu novou dvojici. Hodnota "Package"již byla spárována s "1 min", takže se může do čtvrtého řádku zadat "Formation"a vytvořit tak novou dvojici. To samé se provede pro hodnoty "Coach"a "Package", aby se vytvořily páry s hodnotami "5 min"a "15 min". Následně se doplní hodnoty, které ještě mezi sebou nemají dvojici. Takže "1 min"bude s "Coach", "5 min"s "Package"a "15 min"s "Formation". Pokrok je zobrazen v Tabulce 4.2. [4]

Tabulka 4.2 Testovací tabulka II

Quarter Length	Play Calling
1 min	Package
5 min	Formation
15 min	Coach
1 min	Formation
5 min	Coach
15 min	Package
1 min	Coach
5 min	Package
15 min	Formation

V pátém kroku se potvrdí, že všechny dvojice jsou propojeny:

- 1 min = Package | 1 min = Formation | 1 min = Coach
- 5 min = Formation | 5 min = Coach | 5 min = Package
- 15 min = Coach | 15 min = Package | 15 min = Formation

Pátý krok ověřil, že jsou všechny dvojice zastoupeny a může se tedy přejít znova na krok třetí, který přidá parametr Game Speed a jeho možnosti. (Viz tabulka 4.3) [4].

Tabulka 4.3 Testovací tabulka III

Quarter Length	Play Calling	Game Speed
1 min	Package	Slow
5 min	Formation	Normal
15 min	Coach	Fast
1 min	Formation	
5 min	Coach	
15 min	Package	
1 min	Coach	
5 min	Package	
15 min	Formation	

Ve čtvrtém kroku se pokračuje v doplnění třetího sloupce. Na čtvrtý řádek se přidá možnost "Fast", jelikož zbylé možnosti již mají páry mezi sebou ("Slow"s "1 min"a "Normal"s "Formation"). Doplní se podle řady zbylé možnosti a pokračuje se sedmým řádkem, kde se přidá hodnota "Normal", pak "Fast"a nakonec "Slow"(viz Tabulka 4.4). Nakonec se opět podle pátého kroku zkontrolují dvojice. [4]

Tabulka 4.4 Testovací tabulka IV

Quarter Length	Play Calling	Game Speed
1 min	Package	Slow
5 min	Formation	Normal
15 min	Coach	Fast
1 min	Formation	Fast
5 min	Coach	Slow
15 min	Package	Normal
1 min	Coach	Normal
5 min	Package	Fast
15 min	Formation	Slow

- 1 min = Package = Slow | 1 min = Formation = Fast | 1 min = Coach = Normal
- 5 min = Formation = Normal | 5 min = Coach = Slow | 5 min = Package = Fast

- 15 min = Coach = Fast | 15 min = Package = Normal | 15 min = Formation = Slow

Opět se proces vrátí na třetí krok a přidá se nový parametr "Challenges"kde se přidají hodnoty "Yes"a "No". Pokračuje se na čtvrtý krok. Na třetí řádek se přidá hodnota "Yes"a na čtvrtý hodnota "No". "Yes"v řádku 4 by vytvořilo pouze jednu novou dvojici s "Formation", takže "No"je správná hodnota, která se musí vložit. Řádky 5 a 6 se vyplní hodnotou "No", aby se v každém z těchto řádků rovněž vytvořily dvě nové dvojice. "No"v řádku 7 nevytváří žádné nové dvojice, protože "1 min"je již spárována s "No"v řádku 4 a "Coach"je již spárován s "No"v řádku 6. "Yes"v tomto řádku vytváří novou dvojici s "Normal", takže je to jediná správná volba. Řádky 8 a 9 musí být rovněž vyplněny "Yes", aby se vytvořily nové dvojice s "5 min", resp. s "Formation". Výsledek je viditelný v Tabulce 4.5. [4]

Tabulka 4.5 Testovací tabulka V

Quarter Length	Play Calling	Game Speed	Challenges
1 min	Package	Slow	Yes
5 min	Formation	Normal	No
15 min	Coach	Fast	Yes
1 min	Formation	Fast	No
5 min	Coach	Slow	No
15 min	Package	Normal	No
1 min	Coach	Normal	Yes
5 min	Package	Fast	Yes
15 min	Formation	Slow	Yes

Další postup se v rámci příkladu zrychlí. Po vytvoření kompletního sloupce opět zkontrolují hodnoty a jejich vazby a přejde se zase na krok tři, kdy se přidá parametr "Coach Mode"a vyplní se hodnotami. Začne se vyplňováním hodnot "Yes"a "No". Pouze "Ne"může vytvořit novou dvojici s ostatními čtyřmi hodnotami ve třetím řádku ("15 min", "Coach", "Fast"a "Yes"). Pro řádek 4 se zvolí "Yes", který vytvoří tři nové dvojice (u "No"by byla pouze jedna dvojice s "1 min"). U 5 a 6 řádku se zase vytvoří nové dvojice pouze s hodnotou "Yes". Pro zbylé možnosti je jednou možností hodnota "No". Výsledek je v Tabulce 4.6. [4]

Opět se proces vrátí na 3 krok, přidá se parametr "Multiple Routes"a vloží se první dvě hodnoty "Yes"a "Yes". Pokud by druhá hodnota byla "No", kterákoli z hodnot "Multiple Routes"přidaná do řádku 3 vytvoří čtyři nové dvojice, takže nelze vybrat ani jednu z nich. Hodnota "Yes"vytvoří nové dvojice s hodnotami "15 min", "Coach", "Fast"a "No"(Coach mode), zatímco hodnota "Ne"vytvoří nové dvojice s hodnotami "15 min", "Coach", "Fast"a "Yes"(Challenges). Přejde se tímto na 4 krok a možnosti jsou jasné. Do řádku 3 se vloží hodnota "No"vytvoří nové dvojice ve všech pěti sloupcích

Tabulka 4.6 Testovací tabulka VI

Quarter Length	Play Calling	Game Speed	Challenges	Coach Mode
1 min	Package	Slow	Yes	Yes
5 min	Formation	Normal	No	No
15 min	Coach	Fast	Yes	No
1 min	Formation	Fast	No	Yes
5 min	Coach	Slow	No	Yes
15 min	Package	Normal	No	Yes
1 min	Coach	Normal	Yes	No
5 min	Package	Fast	Yes	No
15 min	Formation	Slow	Yes	No

("Yes"by dalo pouze u čtyř). Čtvrtý, pátý a šestý řádek je opět vhodnější s hodnotou "No", která vytvoří čtyři a dva nové páry. A do posledního tří řádků se doplní možnost "Yes", která vytvoří nový pár. Výsledná Tabulka 4.7 je k dispozici níže. Opět se provede kontrola podle pátého kroku, že vše bylo správně spárováno. [4]

Tabulka 4.7 Testovací tabulka VII

Quarter Length	Play Calling	Game Speed	Challenges	Coach Mode	Multiple Routes
1 min	Package	Slow	Yes	Yes	Yes
5 min	Formation	Normal	No	No	Yes
15 min	Coach	Fast	Yes	No	No
1 min	Formation	Fast	No	Yes	No
5 min	Coach	Slow	No	Yes	No
15 min	Package	Normal	No	Yes	No
1 min	Coach	Normal	Yes	No	Yes
5 min	Package	Fast	Yes	No	Yes
15 min	Formation	Slow	Yes	No	Yes

Vytvořením této kombinatorické tabulky se snížil počet testovacích případů z 207 na 9. V tomto případě ani nebyl využit 6 a 7 krok. Při tomto testování je důležité, aby se nebraly v potaz jen okamžité nebo krátkodobé výsledky testu. Některé z efektů mohou být Započítaly se akce provedené ve hře správně do úspěchů a rekordů sezóny/kariéry? Lze ukládat a načítat relace nebo soubory? Lze správně zahájit a hrát novou relaci? Je možné postoupit do příslušných částí hry nebo příběhu? [4]

Výsledky testu podle tabulky 4.7 mohou být:

- Hra v režimu "Coach"znemožňuje uživateli vybrat nebo ovládat některého z hráčů během hry. Pokud je funkce aktivní, jaký má smysl režim "Multiple Routes", když hráč nemá možnost žádné nastavit.
- Jednou z funkcí NFL 2K5 je poskytování Crib Points za různé úspěchy během hry a jednou z odměn je 10 bodů za každou hodinu hraní. Při hře s délkou čtvrtiny

= 15 minut trvá skutečná herní doba (čtyři čtvrtiny) více než jednu hodinu. Tato odměna může během hraní objevit 2x nebo 3x v závislosti na tempu. Po skončneí hry se ale v přehledu zobrazí tato odměna pouze jednou.

- Tlačítko A (pro XBOX) slouží pro potvrzení voleb. V "Coach"režimu jsou některé akce tlačítka A, například aktivní výběr hráče, blokovány. Vedlejším efektem režimu "Coach Mode"je, že uživatel nemůže přerušit časový limit. Nakonec se čeká přibližně 90 sekund na vypršení časového limitu, než se hra obnoví.

Ekonomika kombinatorických testů Kombinatorické testování může přinést snížení testů až 100000:1. Vše záleží na použitých parametrech a nastavení hodnot. Ovšem některé herní mechanismy jsou důležitější než jiné. V tomto případě je nejvýhodnější provést úplné kombinatorické testování pro kritické funkce a párové pro zbylé. Pro příklad se nastaví rozložení 10 % kritické a 90 % zbylé funkce. Každá z kritických funkcí má cca 100 na sobě (matice 4*4*3*2). Zbývajících 90 % funkcí by mohlo být testováno pomocí párových kombinatorických tabulek a stálo by to pouze 20 testů na funkci. Náklady na úplné kombinatorické testování všech prvků jsou `100*N`, kde N je celkový počet testovaných prvků. Náklady na párové kombinatorické testování 90 % těchto prvků jsou `100*0,1*N + 20*0,9*N = 10*N+18*N = 28*N`. To představuje 72 % úsporu při použití párového testování pro nekritických 90 %. [4, 29]

Další možnost je využití pro tzv. "testy příčetnosti". Počet testů, které se provedou na začátku projektu, zůstane nízký a poté, jakmile hra projde testy příčetnosti, tým se může spolehnout na jiné způsoby provádění "tradičních"nebo "plných"testů. Navíc znalost toho, které kombinace fungují správně, může pomoct vybrat scénáře např. pro propagaci hry. [4, 29]

Souhrn Testy by se měly vytvořit co nejdřív v životním cyklu, aby výsledky byly co nejlevnější a nejefektivnější. Jakmile budou k dispozici návrhové dokumenty nebo storyboardy, měly by se vytvořit kombinatorické tabulky na základě informací, které jsou v té době k dispozici, a využít návrhářů na vytvoření správných scénářů. Zároveň párové kombinované testy poskytují dobrou rovnováhu mezi šířkou a hloubkou pokrytí, což umožní otestovat více oblastí hry, než kdyby se soustředily zdroje jen na několik oblastí. [4, 29]

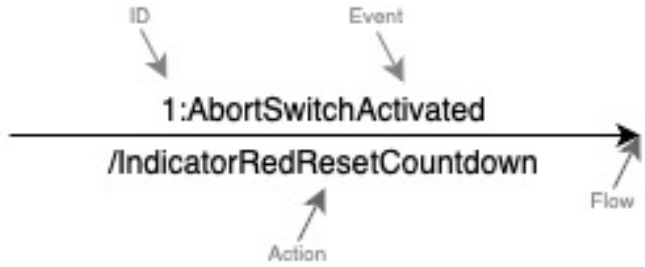
4.3.2 Testovací vývojové diagramy

Testovací vývojové diagramy (TVD) jsou grafické modely, které představují herní chování z pohledu hráče. Tester prochází diagram a prochází známé i neznámé způsoby. TVD podporuje modularitu a úplnost. Díky tomu poskytují formální přístup k návrhu

testů. Testy mohou být opětovně použitelné a dává testerům, vývojářům a výrobcům možnost snadno kontrolovat, analyzovat a poskytovat zpětnou vazbu k návrhům testů. [4, 30]

TVD se skládá z elementů, které jsou nakresleny, označeny a propojeny podle určitých pravidel. Díky něm jsou testy srozumitelné v rámci celé organizace a je možné jejich opětovné použití v budoucích projektech. [4, 30]

Flow je nakreslena jako čára, která spojuje jeden herní stav s druhým. Každé flow má jedinečné ID, jednu událost a jednu akci. Během testování se provede to, co je specifikováno událostí, a poté se zkontrolujete chování specifikované akce s cílovým stavem flow. Na Obrázku 4.7 lze vidět rozbor flow komponenty. [4]



Obrázek 4.7 Flow komponenty

Event (Událost) je operace, kterou vykonal hráč, síť, zařízení nebo interní herní mechanismus. Příkladem může být seslání kouzla, zahájení dialogu nebo výběr zbraně z inventáře. TVD nemusí obsahovat všechny kombinace událostí. Jedno Flow může mít pouze jednu událost, ale jedna událost může představovat více událostí. Název události může objevit v TVD vícekrát, pokud stejná instance nese vždy stejný název. Událost může (ale i nemusí) způsobit přechod do nového herního stavu. Pro vytvoření nové události je potřeba počítat se třemi faktory: [4]

1. Pravděpodobné iterace s dalšími událostmi.
2. Unikátní nebo důležité chování spojené s danou událostí.
3. Unikátní nebo důležité herní stavy, které jsou důsledkem události.

Actions (Akce) Je to dočasné nebo přechodné chování v reakci na událost. Lze vnímat prostřednictvím lidských smyslů a zařízení herní platformy (zvuky, vizuální efekty, zpětná vazba herního ovladače, informace ze sítě atp.). Mohou být detekovány pouze v okamžiku zahájení. Ve flow může být pouze jedna akce, ale více operací může být reprezentováno jednou akcí. Pokud každá instance obsahuje stejný název, v TVD může být název akce vícekrát. [4]

States (Stavy) Je to opakované chování hry, kdy dokud stav neopustíte (nebo se k němu vrátíme), tester bude pozorovat stejné výsledky. Stav je zobrazen jako "bublina"s

textem vevnitř. Každý stav má alespoň jedno vstupní a výstupní flow. Jestli se stejné chování objevuje ve více než jednom stavu v diagramu, může se jednat o stejný stav. V tom případě se musí odstranit duplicity a přepojit flow, aby dávalo smysl. [4]

Primitives (Primitiva) Primitivy se označují události, akce a stavy. Popisují podrobné chování, aniž by zahlcovaly TVD. Seznam primitivů a jejich definic se nachází v datovém slovníku. [4]

Terminators (Terminátoři) Jedná se o políčka, kde je napsáno kde test začíná a kde končí. Každý TVD musí obsahovat vždy 2 terminátory. První je "IN". který představuje vstup do flow a druhý je "OUT", který představuje jeho výstup. [4]

Vytvoření TVD Pro vytvoření nového TVD je potřeba tří fází: příprava, přidělení a konstrukce.

Nejdříve se shromáždí a identifikuj požadavky, které spadají do rozsahu plánovaného testování. Patří zde návrhové dokumenty, storyboardy, demonstrační obrázky, požadavky na SW nebo například předchozí tituly. [4, 30]

V přidělení se odhadne počet TVD a zmapují se k něm herní prvky. Povede se rozdělení velkých sad na malé a pokryjí se související požadavky ve stejném designu. První způsob, jak k tomu přistoupit je otestovat různé "schopnosti"ve hře (léčení, seslání kouzla, zvednutí zbraně) a ke každé variantě (různá kouzla, různé zbraně) vytvořit TVD. Druhým způsobem může být zmapování scénářů k jednotlivým TVD, které se pak specializují na jednotlivé "úspěchy". Těmi mohou být jednotlivé mise, zápasy nebo například výzvy, které v případě splnění značí, že jsou dané "úspěchy"dosažitelné. Založení na "úspěších"by mohlo být použito buď místo přístupu založeného na schopnostech, nebo jako doplněk k němu. Vždy je lepší vytvořit více menších TVL než méně velkých. [4, 30]

U konstrukce má TVD představovat testerovu interpretaci toho, co očekává, že se stane, když se prochází hra od začátku po konec všech stavů hry znázorněných na diagramu. Nemá být založena na opravdovém SW designu. Podle těchto kroků by se mělo TVD vytvářet: [4]

1. Vytvoří se soubor s jedinečným názvem pro TVD.
2. Vloží se obdélník s textem "IN".
3. Vloží se kruh s názvem stavu.
4. Z obdélníku se nakreslí flow ke stavu a vloží se název (ID se vypíše až na konec). Diagram se sestaví tak, jak se prochází testovacím scénářem. Vytváření by mělo být iterativní a dynamické.

5. Od prvního stavu se pokračuje dalším stavem nebo flow. Flow lze připojit zpět k výchozímu stavu, aby bylo možné je připojit k požadované chování, které je přechodné nebo chybí.

6. Každou návaznost flow na jeden nebo víc požadavků/možností se zaznamenává.

7. Pro každé flow z A do B zkontrolujte možnosti na různé přechody a podle potřeby je případně upravte.

8. Jakmile je diagram dokončen, musí se zkontrolovat, jestli existují alternativní cesty. Je potřeba jej ověřit s dokumentací a případně jej upravit nebo zdokumentovat.

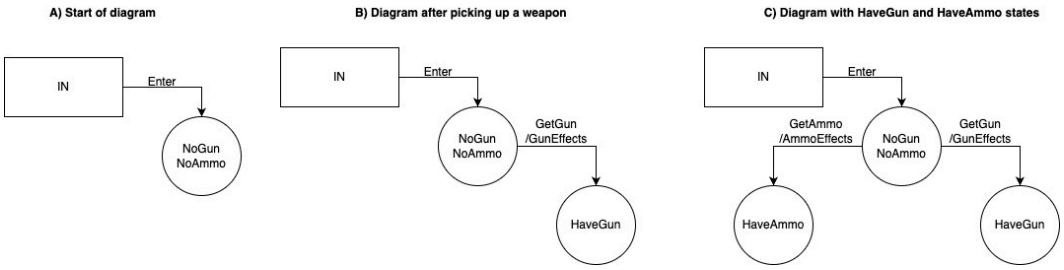
9. Vytvoří se obdélník s textem "OUT".

10. Vyberou se stavy, které s tímto obdélníkem mají být spojeny. Musí se vzít v potaz místa, která jsou vhodná k ukončení testu a případného následného pokračování testu nového. Do flow směřujícím do obdélníku s "OUT"se dá událost "Exit". Každý stav může mít maximálně jedno takové flow.

11. Aktualizuje se název v obdélnících "IN"a "OUT"na "IN_xxx"a "OUT_xxx", kdy "xxx"je popis TVD.

12. Nakonec se očíslují všechny flow za pomocí ID.

Následuje příklad založený na možnosti vzít zbraň a munici ve hře Unreal Tournament 2004 (Hra již správně sleduje počet nábojů a provádí správné zvukové a vizuální efekty.). I když se může zdát tato schopnost z pohledu první osoby jako malá funkcionalita, tak první čtyři záplaty (pět závad souvisejících s municí + čtrnáct chyb souvisejících se zbraněmi) ve hře patřily právě tomuto systému. Diagram se započne přímo bodem ve hře, kterým se má začít. Všechny TVD diagramy začínají IN obdélníkem, ze kterého jde flow do prvního stavu, kterého se má dosáhnout. [4, 30]

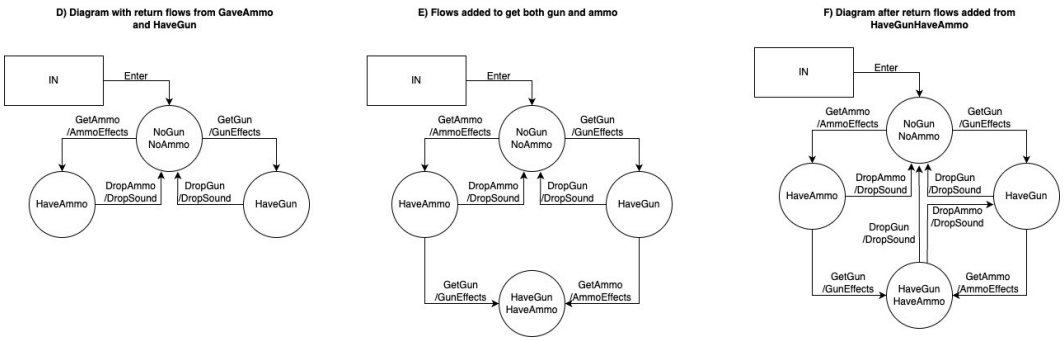


Obrázek 4.8 TVD diagram I

Začíná se situací, kdy hráč nemá žádnou zbraň ani náboje. Z "IN"obdélníku se vytvoří flow do stavu `NoGunNoAmmo`. Název události je "Enter"(ID se uvede až na konci). Hotový první krok je na Obrázku 4.8 v části A. [4]

Následuje vytvoření reakcí na tuto akci. Jedna z možností je nalezení zbraně a její sebrání. Vytvoří se tedy nový stav `HaveGun` a v něm se připojí nové flow. Dočasné efekty jsou reprezentovány akcí ve flow. Ke flow se tedy dá název akce `GunEffect` a událost se pojmenuje `GetGun`. Výsledek je viditelný na Obrázku 4.8 v části B. [4]

Pokud hráč získá munici dříve než samotnou zbraň, přidá se další stav `HaveAmmo`. Nový stav se propojí se stavem `NoGunNoAmmo` za pomoci flow s událostí `GetAmmo` a akcí `AmmoEffect`. Schéma reprezentuje Obrázek 4.8 část C. [4]



Obrázek 4.9 TVD diagram II

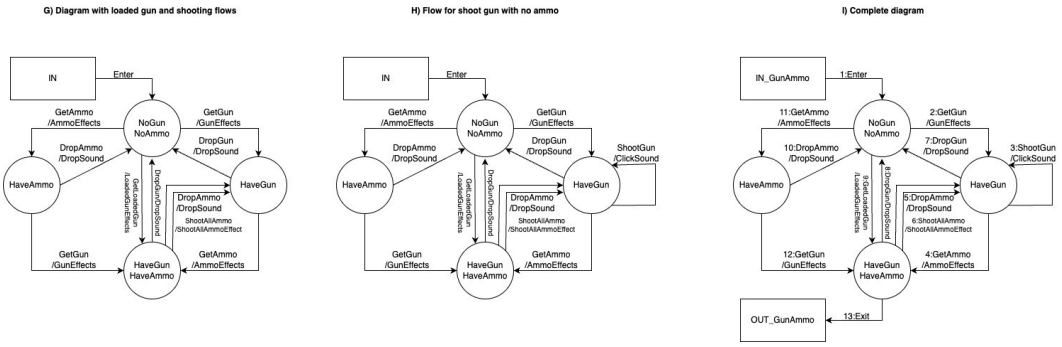
K vytvořeným stavům je dobré vytvořit navazující flows pro vrácení k původnímu stavu. Do stavu `NoGunNoAmmo` je možné se vrátit odhozením munice nebo zbraně. Pokud existuje více způsobů, každý z nich by měl být vyjádřen v TVD. Zpětné flows jsou znázorněny na Obrázku 4.8 v části D. Oba obsahují stejnou akci `DropSound`, akorát se liší událostí (buď `DropAmmo` nebo `DropGun`). [4]

Jakmile diagram obsahuje jak stav pro vlastnění zbraně, tak pro vlastnění nábojů, je potřeba vytvořit nový stav, který obsahuje obojí. Vytvoří se tedy `HaveGunHaveAmmo` a jsou k němu napojeny flows jak z `HaveAmmo`, tak i `HaveGun`. Flows i s novým stavem je znázorněno na Obrázku 4.9 v části E. [4]

Tento test je založen na tom, že se munice automaticky načte, když máte odpovídající zbraň. Pro stav `HaveGunHaveAmmo` se vytvoří návratové flows s událostmi `DropGun` a `DropAmmo`. K oběma událostem je akce `DropSound`. Výsledný diagram je k vidění na Obrázku 4.9 v části F. [4]

Udělá se validace, jestli je TVL kompletní nebo je možné přidat další stavy k nábojům nebo zbraním. Začne se od nejvzdálenějšího stavu a jde se po proudu směrem nahoru. Střelbou se spotřebovává munice, dokud se opět nedostane do `HaveGun`. Jelikož jsou oba stavy zapojené do správného flow, tak stačí přidat nové flow z `HaveGunHaveAmmo` do `HaveGun`. Zároveň může nastat situace, kdy hráč získá zbraň a zároveň i náboje. Vznikne tím tedy flow z `NoGunNoAmmo` do `HaveGunHaveAmmo`. Diagram je k vidění na Obrázku 4.10 v části G. [4]

`ShootAllAmmo` způsobí zvuky, grafické efekty a poškození jiného hráče nebo pro-



Obrázek 4.10 TVD diagram III

středí. `GetLoadedGun` způsobí efekty podobné sebrání nenabité zbraně a její munice. Akce se pojmenovaly `AllAmmoEffects` a `LoadedGunEffect`. Událost `ShootAllAmmo` ukazuje, že testovací události nemusí být atomické. TVD nemusí potřebovat samostatnou událost a flow pro jednotlivé náboje (pokud to není potřeba). Pro `HaveGun` a `HaveAmmo` se udělá totéž, co pro `HaveGunHaveAmmo`. Ze zbraně je možné se pokusit vystřelit, i když nemá žádné náboje. Z toho důvodu se vytvoří další flow, které bude odkazovat zpět na `HaveAmmo`. Aktualizovaný diagram je na Obrázku 4.10 v části H. [4]

Nyní už zbývá dodělat jen obdélník s textací "OUT"a očíslovat jednotlivé flows (každý musí mít jedinečné ID). Obdélníky "IN"a "OUT"získají pojmenování podle testované funkce TVD. Názvem se pak může lépe identifikovat TVD ve sbírce TVD, která reprezentuje více testovacích scénářů. Díky názvu je také možné definovat další detaily v datovém slovníku. Tím je celý diagram dokončen. K vidění je na Obrázku 4.10 v části I. [4]

Datový slovník V datovém slovníku jsou podrobné popisy všech jasně identifikovatelných základních prvků v TVD. Pokud se použije název základního prvku v různých částech TVD, má tento název vždy stejný význam a odkazuje na stejnou definici. Datový slovník lze opakovaně využívat v různých TVD. Pokud se jako příklad vezme Obrázek 4.10 (část I), tak ten odkazuje na zbraně a náboje. Většina her zahrnujících zbraně poskytuje více typů zbraní a munice, která je pro každou z nich specifická. Může se tedy pro každou zbraň vytvořit vlastní kopie TVD s jejich přizpůsobením nebo použít jeden obecný TVD diagram a za pomoci datového slovníku interpretovat různá specifika zbraní. [4, 30]

Pro příklad se vytvoří datový slovník vytvořeného diagramu výše. Události se zapisují normálně. Mohou se přidávat obrázky pro lepší vizualizaci. Akce a stavy se zapisují jako odrážky, které, které se tímto od událostí vizuálně rozdělují (pro zpřehlednění je dobré je psát s checkboxy). V seznamu níže jsou některé primitivy pro zbraň Bio-Rifle, kdy jsou seřazeny v abecedním pořadí. [4]

Datový slovník z příkladu Obrázku 4.10 (část I):

`AmmoEffects`

☐ Zkontroluje se, jestli se ozve zvuk nábojů do Bio-Rifle.

☐ Zkontroluje se, zda se ve hře dočasně zobrazí bílý text "You picked up some Bio-Rifle ammo"v dolní části obrazovky.

☐ Zkontroluje se, jestli dočasný text na displeji pomalu mizí.

`DropGun`

Stisknutím klávesy "\"upustíte vybranou zbraň.

`DropSound`

☐ Zkontroluje se, zda je vydán zvuk pádu položky.

`Enter`

Vybere se zápas a kliknutím na tlačítko FIRE se spustí.

`Exit`

Stisknutím klávesy "ESC"se ukončí zápas.

`GetAmmo`

V aréně se najde na podlaze balíček s municí pro Bio-Rifle a přejde se přes něj.

`GetGun`

Najde se NEnabitá Bio-Rifle vznášející se nad podlahou arény a přejde se přes ní.

`GetLoadedGun`

Najde se nabitá Bio-Rifle vznášející se nad podlahou arény a přejde se přes ní.

`GunEffects`

☐ Zkontroluje se, zda vydává správný zvuk pro Bio-Rifle.

☐ Zkontroluje se, jestli se ve hře dočasně zobrazí bílý text "You got the Bio-Rifle"nad ikonami zbraní v dolní části obrazovky.

☐ Zkontroluje se, zda se ve hře současně zobrazuje dočasně nápis "Bio-Rifle"s modrým textem nad hlášením "You got Bio-Rifle".

☐ Zkontroluje se, jeslti dočasné texty na displeji pomalu mizí.

`HaveAmmo`

☐ Zkontroluje se, zda je ikona Bio-Rifle v grafickém inventáři zbraní v dolní části obrazovky prázdná.

☐ Zkontroluje se, jestli se hlaveň biologické pušky nezobrazuje před postavou.

☐ Zkontroluje se, zda nelze vybrat zbraň Bio-Rifle pomocí kolečka myši.

☐ Zkontroluje se, jestli se nezměnil zaměřovací terč ve středu obrazovky.

`HaveGun`

☐ Zkontroluje se, zda je v grafickém inventáři zbraní v dolní části obrazovky přítomna ikona Bio-Rifle.

☐ Zkontroluje se, jestli je hlaveň Bio-Rifle zobrazena před herní postavou.

☐ Zkontroluje se, zda se může vybrat zbraň Bio-Rifle pomocí kolečka myši.

☐ Zkontroluje se, jestli se zaměřovací terč pušky Bio-Rifle zobrazuje jako malý modrý lomený trojúhelník uprostřed obrazovky.

☐ Zkontroluje se, jestli je počet nábojů v pravém rohu obrazovky roven 0.

`HaveGunHaveAmmo`

☐ Zkontroluje se, zda je v grafickém inventáři zbraní v dolní části obrazovky přítomna ikona Bio-Rifle.

☐ Zkontroluje se, jestli je hlaveň Bio-Rifle zobrazena před herní postavou.

☐ Zkontroluje se, zda se může vybrat zbraň Bio-Rifle pomocí kolečka myši.

☐ Zkontroluje se, jestli se zaměřovací terč pušky Bio-Rifle zobrazuje jako malý modrý lomený trojúhelník uprostřed obrazovky.

☐ Zkontroluje se, jestli je počet nábojů v pravém rohu obrazovky roven 40.

`IN_GunAmmo`

Spuštění Unreal Tournament 2004 na testovacím PC.

`LoadedGunEffects`

☐ Zkontroluje se, zda vydává správný zvuk pro Bio-Rifle.

☐ Zkontroluje se, jestli se ve hře dočasně zobrazí bílý text "You got the Bio-Rifle"nad ikonami zbraní v dolní části obrazovky.

☐ Zkontroluje se, zda se ve hře současně zobrazuje dočasně nápis "Bio-Rifle"s modrým textem nad hlášením "You got Bio-Rifle".

☐ Zkontroluje se, jestli dočasné texty na displeji pomalu mizí.

`NoGunNoAmmo`

☐ Zkontroluje se, zda je ikona Bio-Rifle v grafickém inventáři zbraní v dolní části obrazovky prázdná.

☐ Zkontroluje se, jestli se hlaveň Bio-Rifle nezobrazuje před herní postavou.

☐ Zkontroluje se, zda nelze vybrat zbraň Bio-Rifle pomocí kolečka myši.

`OUT_GunAmmo`

V hlavním menu se klikne na "Exit UT2004"pro ukončení hry.

TVD cesty Testovací trasa je řada flows, které se prochází ve specifickém pořadí a vytváří tímto testovací scénáře. Cesty začínají v "IN"a končí ve stavu "OUT". Prováděním cest se sledují události, akce a stavy v TVD. Textový skript lze sestavit vystřižením a vložením primitiv do pořadí, v jakém se na cestě vyskytují. Testeři podle tohoto skriptu provedou každý test a podrobnosti o jednotlivých primitivech najdou v datovém slovníku. Automatizované skripty se vytvářejí stejným způsobem, jen se místo textových instrukcí pro lidského testera vkládají řádky kódu. Testy mohou být prováděny podle jediné strategie po celou dobu trvání projektu nebo se sady cest mohou měnit podle vyspělosti kódu hry při jejím postupu různými milníky. [4, 30]

Cílem *generování minimální cesty* je vytvoření co nejméně cest s pokrytím všech flows z diagramu ("pokrytí"znamená, že flow je v testu použit alespoň jednou). Výhoda je v nízký počet testů. Na druhou stranu se získávají dlouhé cesty, které mohou zabránit v testování některých částí diagramu až do pozdější fáze projektu, když se něco pokazí na začátku testovací cesty. Následuje příklad z Obrázku 4.10 (část I).

Začne se u IN a přejde se z flow 1 do `NoGunNoAmmo`, poté s flow 2 do `HaveGun`. Protože flow 3 se smyčkou vrací do `HaveGun`, přejde se na něj a poté se opustí flow 4 s `HaveGun`. Minimální cesta je zatím **1, 2, 3, 4** (jedná se tedy o minimální cestu). Nyní se z `HaveGunHaveAmmo` vratí do `HaveGun` přes flow 5. Protože flow 6 také vede z `HaveGunHaveAmmo` do `HaveGun`, vezme se opět flow 4 a tentokrát se použije flow 6 pro návrat do `HaveGun`. V této fázi je minimální cesta **1, 2, 3, 4, 5, 4, 6**. Flow 7 se vyjme z `HaveGun` a vrátí se zpět do `NoGunNoAmmo`. Teď se díky flow 9 dostane do `HaveGunHaveAmmo`, kde se vrátí zpět po flow 8. Nyní je cesta: **1, 2, 3, 4, 5, 4, 6, 7, 9, 8**. Z `NoGunNoAmmo` se využije flow 11 k `HaveAmmo` a vrátí se zpět do `NoGunNoAmmo` za pomoci flow 10. Přes flow 11 se dostane opět do `HaveAmmo`, kde se přes flow 12 přejde do `HaveGunHaveAmmo` a nakonec přes flow 13 do pole OUT. Dokončená minimální cesta je **1, 2, 3, 4, 5, 4, 6, 7, 9, 8, 11, 10, 11, 12, 13**. Všech třináct toků je tedy TVD

pokryto patnácti testovacími kroky. Pro danou TVD obvykle existuje většinou více než jedna "správná"minimální cesta. Například **1, 11, 10, 11, 12, 8, 9, 5, 7, 2, 3, 4, 6, 4, 13** je také minimální cesta. [4]

Generování základní cesty je definováno jako co nejkratší cesta z IN do OUT, která prochází co největším počtem stavů bez opakování nebo smyčky zpět. Z této základní cesty se pak vytváří další cesty, které se, pokud možno mění nebo vracejí k základní cestě a sledují ji, aby dosáhly pole OUT. Takto proces pokračuje, dokud nejsou všechny flows v diagramu použity alespoň jednou. Základní cesty jsou obsáhlejší než minimální cesty, ale stále úspornější než snaha pokrýt všechny možné cesty diagramem. Jednou z nevýhod základních cest je zvýšené úsilí při generování a provádění cest oproti použití přístupu minimálních cest. [4]

Vytvoří se základní cesta, která je **1, 2, 4, 13**. Následuje vytvoření "odvozených"cest z flow 1. Flow 2 je již v základní linii použit, takže se vezme flow 9 na `HaveGunHaveAmmo`. Odtud se přes flow 8 vrátí zpět na základní cestu. Zbytek základní linie sledujte podél flow 2, 4 a 13. První odvozená cesta z flow 1 je **1, 9, 8, 2, 4, 13**. Flow 11 vychází z `NoGunNoAmmo`, který doteď nebyl použit, takže se tím následuje k `HaveAmmo`. Pak přes flow 10 pro návrat k základní linii. Touto cestou se dokončí sledováním zbytku základní linie až k poli OUT. Tato druhá cesta odvozená od flow 1 je **1, 11, 10, 2, 4, 13**. Následuje vytvoření odvozené cesty z flow 2, která je **2 je 1, 2, 7, 2, 4, 13**. Flow 4 přivádí testera k `HaveGunHaveAmmo` z něhož vycházejí tři flows, které nejsou na základní linii: 5, 6 a 8. Flow 8 již byl využit, takže není potřeba ho použít znova. Flow 5 a 6 se začlení do základní linie stejným způsobem, protože oba se vracejí do stavu `HaveGun`. Odvozená cesta využívající flow 5 je **1, 2, 4, 5, 4, 13** a odvozená cesta pro flow 6 je **1, 2, 4, 6, 4, 13**. I když jsou ze základní cesty odvozeny cesty následující, pořád zbývá flow 12, které nebylo nikde využito. Přes flow 1 a 11 se dostane tester do stavu `HaveAmmo` a využije se flow 12. Pro cestu zpět `HaveGunHaveAmmo` se využije flow 8, kde se dostane do `NoGunNoAmmo` a nakonec se vydá po zbytku základní linie. Tato závěrečná cesta je **1, 11, 12, 8, 2, 4, 13**. Souhrn je Tabulce 4.8. [4]

Tabulka 4.8 Souhrn základní cesty

Základní linie	Z flow 1	Z flow 2	Z flow 4	Z flow 11
1, 2, 4, 13	1, 9, 8, 2, 4, 13	1, 2, 3, 4, 13	1, 2, 4, 5, 4, 13	1, 11, 12, 8, 2, 4, 13
	1, 11, 10, 2, 4, 13	1, 2, 7, 2, 4, 13	1, 2, 4, 6, 4, 13	

Odborně konstruované cesty jsou jednoduše cesty, které vytvořil "expert"na testy nebo funkce na základě svých znalostí o tom, jak je pravděpodobné, že funkce selže, nebo kde potřebuje získat jistotu v určité sadě chování. Lze je využít jak se základním, tak s minimální strategií, kdy není ani potřeba pokrýt všechny flows, jen je pořád důležité, aby vedly od pole IN do pole OUT. Odborné cesty mohou být efektivní při hledání

problémů, pokud existuje firemní paměť o tom, co se v minulosti nepodařilo nebo jaké nové herní funkce jsou nejcitlivější. Nevýhoda je v nepokrytí všech podstatných flows a zkreslení, které může vzniknout, když testy neobsahují "neočekávané"situace. Příklady strategií mohou být: opakování určitého flow nebo sekvence flows v kombinaci s jinými variantami cesty; vytváření cest, které zdůrazňují neobvyklé nebo zřídka se vyskytující události; vytvářet cesty, které zdůrazňují kritické nebo složité stavy; vytvářet extrémně dlouhé cesty a v případě potřeby opakovat flow; modelujte cesty podle nejběžnějších způsobů, jakými bude funkce používána. [4]

Pro příklad se využije strategie "zdůraznění kritických nebo složitých stavů". V tomto případě bude zdůrazněn stav `HaveGun` (každá cesta bude alespoň jednou procházet tímto stavem). Cílem je také pokrýt všechny flows touto sadou cest. Aby byly cesty krátké, musí se zaměřit na flow "Exit", jakmile byl použit stav `HaveGun`. Jedním ze způsobů je přejít na `HaveGun`, zkusit vystřelit a pak odejít (cesta **1, 2, 3, 4, 13**). Další z cest by mohla zahrnout událost `DropGun` z flow 7, kdy nejkratší cesta veze skrze flow 9 a 13 (cesta **1, 2, 7, 9, 13**). Musí se zahrnout i další flows jdoucí z `HaveGunHaveAmmo` do `HaveGun`. Vzniknou tak cesty **1, 2, 4, 5, 4, 13** a **1, 2, 4, 6, 4, 13**. Následně pomocí flow 8 dokončit opouštění události `HaveGunHaveAmmo` (cesta **1, 2, 4, 8, 9, 13**). Dále může vzniknout cesta **1, 11, 12, 5, 4, 13**. Ta mapuje cestu do `HaveGunHaveAmmo` a nazpět. cesta **1, 2, 4, 5, 4, 13** může být smazána, protože byla nahrazena předešlou cestou se flow 12. Poslední cesta je **1, 11, 10, 2, 4, 13**, kdy zbývalo využít flow 10. Souhrn je Tabulce 4.9. [4]

Tabulka 4.9 Souhrn odborné cesty

Odborné cesty	později nadbytečné cesty
1, 2, 3, 4, 13	
1, 2, 7, 9, 13	
1, 2, 4, 6, 4, 13	
1, 2, 4, 8, 9, 13	
1, 11, 12, 5, 4, 13	1, 2, 4, 5, 4, 13
1, 11, 10, 2, 4, 13	

Kombinace strategií cest:

1. Vytvoří se odborné cesty, které vývojáře nejvíce zajímají, nebo na cesty, které se zaměřují pouze na části hry, které jsou k dispozici pro testování.

2. Použijí se základní cesty, aby se vytvořila určitá důvěra v testovanou funkci. Je možné začít tím, že se vyzkouší, zda hra projde základní cestou, a teprve poté se pokusí použít ostatní cesty v sadě.

3. Jakmile všechny základní cesty projdou, průběžně se používají minimální cesty, aby se mohlo sledovat, zda se funkce nerozbila.

4. Před jakýmkoliv větším předváděním je dobré se vrátí zpět k základním a/nebo expertním cestám.

Následuje vytvoření testovacího případu z cesty **1, 11, 12, 13**, kdy se bude testovat získání munice, poté získání zbraně a následné ukončení. Pro popis tohoto testovacího případu se použije definice datového slovníku. [4]

Spustí se Unreal Tournament 2004 na testovacím počítači.

Vybere se zápas a kliknutím na tlačítko FIRE se spustí.

☐ Zkontroluje se, zda je ikona Bio-Rifle v grafickém inventáři zbraní v dolní části obrazovky prázdná.

☐ Zkontroluje se, jestli se hlaveň Bio-Rifle nezobrazuje před herní postavou.

☐ Zkontroluje se, zda nelze vybrat zbraň Bio-Rifle pomocí kolečka myši.

V aréně se najde na podlaze balíček s municí pro Bio-Rifle a přejde se přes něj.

☐ Zkontroluje se, zda se ozve zvuk munice do pušky Bio-Rifle.

☐ Zkontroluje se, zda je ikona Bio-Rifle v grafickém inventáři zbraní v dolní části obrazovky prázdná.

☐ Zkontroluje se, zda se hlaveň biologické pušky nezobrazuje před herní postavou.

☐ Zkontroluje se, zda nelze vybrat zbraň Bio-Rifle pomocí kolečka myši.

☐ Zkontroluje se, zda se nezměnil zaměřovací terč ve středu obrazovky.

Najde se nenabitá Bio-Rifle vznášející se nad podlahou arény a přejde se přes ní.

☐ Zkontroluje se, zda je vydáván zvuk Bio-Rifle.

☐ Zkontroluje se, zda se ve hře dočasně zobrazuje nápis "You got the Bio-Rifle"bílým textem nad ikonami zbraní v dolní části obrazovky.

☐ Zkontroluje se, zda hra dočasně zobrazuje nápis "Bio-Rifle"modrým textem nad hlášením "You got Bio-Rifle".

☐ Zkontroluje se, zda všechny dočasné texty na displeji pomalu mizí.

☐ Zkontroluje se, zda je v grafickém inventáři zbraní v dolní části obrazovky přítomna ikona Bio-Rifle.

☐ Zkontroluje se, zda je hlaveň biologické pušky vykreslena před herní postavou.

☐ Zkontroluje se, zda se může vybrat zbraň Bio-Rifle pomocí kolečka myši.

☐ Zkontroluje se, zda zaměřovací terč pušky Bio-Rifle zobrazuje jako malý modrý lomený trojúhelník uprostřed obrazovky.

☐ Zkontroluje se, zda je počet nábojů v pravém rohu obrazovky 40.

Stiskne se klávesa ESC a ukončí se zápas.

V hlavní nabídce se klikne na "EXIT UT2004"a ukončí se hra.

Shrnutí TVD slouží k vytvoření modelů toho, jak by hra měla fungovat z pohledu hráče. Testy prokáží, zda dojde k očekávanému chování, a naopak nedojde k neočekávanému chování. Složité funkce mohou být reprezentovány komplexními TVD, ale upřednostňuje se řada menších. V tabulce 4.10 jsou uvedeny některé pokyny pro volbu mezi použitím kombinatorické tabulky nebo TVD. Pokud má funkce nebo scénář atributy, které spadají do obou kategorií, zváží se provedení samostatných návrhů pro každý typ. Také pro vše, co je pro úspěch hry kritické, se vytvoří testy pomocí obou metod, pokud je to možné. [4, 30]

Tabulka 4.10 Výběr metodiky návrhu testů

Atribut	Kombinatorické test.	TVD
Nastavení hry	✓	
Možnosti hry	✓	
Konfigurace hardwaru	✓	
Přechody mezi herními stavy		✓
Opakovatelné funkce		✓
Souběžné stavy	✓	
Provozní tok		✓
Paralelní volby	✓	✓
Herní cesty / trasy		✓

4.3.3 Cleanroom testování

Odvíjí se z Cleanroom Software Engineering. Původním účelem testování v Cleanroom bylo procvičení softwaru za účelem měření střední doby do selhání (MTTF - mean time to failure) v průběhu projektu. Zde bude testování zaměřeno na problém, proč zákazníci nacházejí problémy ve hrách poté, co prošly tisíci hodinami testování před vydáním. Uživatelé nacházejí chyby v softwaru tím, že jej používají tak, jak jej používají. Cleanroom vytváří testy, které hrají hru tak, jak ji budou hrát hráči. [4, 31]

Pravděpodobnosti použití (také jako frekvence použití), se využívá ke zjištění, jak často by měly být herní funkce používány, aby realisticky napodobovaly způsob, jakým

budou hráči hru používat. Vychází se jak z různých výzkumů, tak i vlastních očekávání. Vzít v potaz se musí i vývoj hráče v průběhu hry. Vzorce chování hráče by mohou být jiné těsně po spuštění tutoriálu než v době, kdy hráč dosáhne závěrečné úrovně. Například zápasy nebo závody by na konci hry mohou trvat déle kvůli vyšší obtížnosti a bližšímu přizpůsobení dovedností hráče herním protivníkům. Informace o používání lze definovat třemi různými způsoby: [4, 31]

- Použití v závislosti na režimu.
- Použití podle typu hráče.
- Použití v reálném životě.

Používání na základě režimu má vliv na to, jaký režim hráč používá. Například hra pro jednoho hráče, kampaň, multiplayer nebo online. Hra pro jednoho hráče může obsahovat menší počet akčních pasáží nebo misí, než v online režimu či multiplayeru. Omezení může být též ve výběru rasy, klanu, zbraní, vozidel nebo oblastí, kde se hra odehrává. [4, 31]

Kampaně obvykle začínají se základním vybavením a protivníky a v průběhu se objevují složitější prvky. Sportovní hry poskytují režimy Franchise nebo Season, které nabízí možnosti typu tréninkový kemp, obchodování s hráči, vyjednávání o platech atp. RPG hry poskytnou silnější kouzla, brnění, zbraně a protivníky s tím, jak se herní postavě zvyšuje úroveň. [4, 31]

Multiplayer může probíhat jak na jednom počítači, tak na propojených konzolích nebo přes internetový server. Zároveň hráči mají často možnost komunikovat přímo v samotné hře (přes chat nebo voláním) a někdy i komunikují napříč různými národnostmi a časovými osami. I to může mít vliv na testování, kdy je dobré se zaměřit na různé jazykové verze nebo vytíženost serverů. [4, 31]

Použití podle typu hráče je odvozeno od Richard A. Bartle (kniha Hearts, Clubs, Diamonds, Spades: Players Who Suit MUDs), který definoval, že hráči kladou důraz na achievement (úspěchy), průzkum, socializaci nebo zabíjení. [4, 31]

Hráč zaměřený na achievement (úspěch) chce plnit herní cíle, mise a úkoly. Uspokojení mu přinese co nejefektivnější postup na vyšší úrovni, počet bodů a peněz, které má jeho postava. Tento typ hráče může hrát na těžší obtížnosti nebo si ji sám stěžuje (například jde do boje pouze s nožem nebo vybírá pouze negativní komentáře). Zajímat je bude také dosahování bonusových cílů a plnění bonusových misí. [4, 31]

Průzkumníci chtějí zjistit, co hra nabízí. Budou cestovat a hledat neznámá místa a okraje mapy. Jejich pozornost upoutá nezmapované území, pokusí se otevřít všechny dveře a zkontrolovat zásoby ve všech obchodech. [4, 31]

Hráči, kteří touží po socializaci využívají hru jako prostředek ke hraní rolí a poznávání ostatních hráčů. Je pro ně důležité mít možnost komunikace a zapojení do sociálních skupin ve hře, jako jsou klany, cechy apod. Socializátoři budou využívat speciální herní funkce, jakmile se o nich dozvědí od ostatních hráčů. [4, 31]

Hráči s oblibou v zabíjení si užívají, když mohou z ostatních hráčů dostat to nejlepší. Zapojují se do bitev hráč versus hráč a hráč versus bot. Využívají různých komunikačních nástroje ve hře, aby nalákali a ponížili své soupeře. [4, 31]

Krom těchto čtyř typů existují i další, které je dobré vzít v potaz: [4]

- Příležitostný hráč: Drží se převážně funkcí popsaných v tutoriálu, uživatelské příručce a uživatelském rozhraní na obrazovce.
- Hard-core hráč: Používá funkční klávesy, makra, turbo tlačítka a speciální vstupní zařízení, jako jsou joysticky a volanty. Hledá triky a tipy na internetu.
- Button Masher: Přednost rychlosti a opakování před opatrností a obranou.
- Přizpůsobitel: Využívá všechny funkce přizpůsobení hry a hraje hru s vlastními prvky. Bude také obsahovat odemčené předměty, obtisky, dresy, týmy atd.
- Vykořisťovatel: Vždy hledá zkratku. Používá cheatovací kódy, hledá trhliny ve zdech zóny a vybírá protivníky z tajných nebo nedostupných míst. Vytváří boty k výrobě předmětů, získávání bodů a zvyšování úrovně.

Použití podle reálného života využívá mechanismus pro zachycení preferencí hráče a zviditelnění těchto informací. Stejně tak lze ukládat tendence trenérů a slavných hráčů a dodávat je spolu se hrou, aby si hráči mohli vyzkoušet vlastní strategie proti jejich jedinečným stylům hry. Dříve zmíněná hra ESPN NFL 2K5 má takovou funkci zvanou VIP profil. Tyto tendence je dobré zohlednit při testování založeném výhradně na vyváženém používání herních funkcí. Pokud by se takto netestovalo, neodhalilo by to vady, jako je např. přetečení paměti způsobené opakovaným klepáním na tlačítko A. [4, 31]

Generování Cleanroom testů je možné pomocí kterékoli z metod z této DP. Každému kroku testu je třeba přiřadit pravděpodobnost použití, kde se vyberou kroky testu, hodnoty nebo větve a seřadí se za sebou tak, aby vytvořily testy, které odrážejí použití testera. Například pokud se očekává, že hráč simulační hry bude v 50 % případů rozvíjet obytné nemovitosti, ve 30 % případů komerční nemovitosti a ve 20 % případů průmyslové nemovitosti, pak testy Cleanroom budou odrážet stejné četnosti. [4, 31]

Kombinatorické Cleanroom testy Počet testů, které mají být vytvořeny, určí tester a hodnoty pro každý test budou vybrány spíše na základě četnosti jejich použití než na základě toho, zda splňují jednu nebo více potřebných dvojic hodnot. Do tabulky se vloží parametr pravděpodobnosti použití, kdy soubor hodnot přiřazených k jednomu parametru musí dávat dohromady 100 %. [4]

- Příležitostný hráč: Drží se převážně funkcí popsaných v tutoriálu, uživatelské příručce a uživatelském rozhraní na obrazovce.
- Hard-core hráč: Používá funkční klávesy, makra, turbo tlačítka a speciální vstupní zařízení, jako jsou joysticky a volanty. Hledá triky a tipy na internetu.
- Button Masher: Přednost rychlosti a opakování před opatrností a obranou.
- Přizpůsobitel: Využívá všechny funkce přizpůsobení hry a hraje hru s vlastními prvky. Bude také obsahovat odemčené předměty, obtisky, dresy, týmy atd.

Příklad bude ze hry HALO a jejího rozšířeného nastavení. Hodnoty z menu jsou: [4]

- Look Sensitivity: 1, 3 (default), 10
- Invert Thumbstick: Yes, No (default)
- Controller Vibration: Yes (default), No
- Invert Flight Control: Yes, No (default)
- Auto Center: Yes, No (default)

A) Look Sensitivity values with usage percentages

Look Sensitivity	Casual	Achiever	Explorer	Multiplayer
1	10	0	10	5
3	85	75	70	75
10	5	25	20	20
TOTAL	100	100	100	100

B) Invert Thumbstick values with usage percentages

Invert Thumbstick	Casual	Achiever	Explorer	Multiplayer
YES	10	40	30	50
NO	90	60	70	50
TOTAL	100	100	100	100

C) Controller Vibration values with usage percentages

Controller Vibration	Casual	Achiever	Explorer	Multiplayer
YES	80	75	50	90
NO	20	25	50	10
TOTAL	100	100	100	100

D) Invert Flight Control values with usage percentages

Invert Flight Control	Casual	Achiever	Explorer	Multiplayer
YES	25	60	50	90
NO	75	40	50	10
TOTAL	100	100	100	100

E) Auto-Center values with usage percentages

Auto-Center	Casual	Achiever	Explorer	Multiplayer
YES	30	0	20	10
NO	70	100	80	90
TOTAL	100	100	100	100

Obrázek 4.11 Příklady tabulek ke Cleanroom testování

Všechna data se budou brát z Obrázku 4.11, který obsahuje potřebné tabulky. Hodnoty do tabulky se mohou vymyslet nebo použít nějaký generátor náhodných čísel.

Neexistuje žádný špatný způsob, pokud výběr čísel není zaujatý vůči nějakému rozsahu čísel. V tomto příkladu se vytvoří tabulka pro příležitostného hráče. [4]

Protože je k dispozici pět parametrů, získá se pět náhodných čísel v rozsahu 1-100. Ta budou postupně použita k určení hodnot jednotlivých parametrů v prvním testu. [4]

Pro jednotlivé testy se vygenerují náhodná čísla:

- První: 30, 89, 77, 25 a 13
- Druhý: 79, 82, 27, 8, a 57
- Třetí: 32, 6, 64, 66, a 11
- čtvrtý: 86, 64, 95, 50, a 22
- Pátý: 33, 21, 63, 85, a 76
- Šestý: 96, 36, 18, 48, a 12

Tabulka 4.11 Hotová Cleanroom kombinatorická tabulka pro příležitostného hráče

Test	Look Sens.	Invert Thumbstick	Controller Vib.	Invert F. Control	Auto-center
1	3	No	Yes	Yes	Yes
2	3	No	Yes	Yes	No
3	3	Yes	No	No	Yes
4	3	No	Yes	No	Yes
5	3	No	Yes	No	No
6	10	No	Yes	No	Yes

Pro ukázku, jak tabulku sestrojit se vezme první test. Na Obrázku 4.11 v sekci A je dáno, že příležitostný hráč nastaví Look sens. na 1 v 10 % případů, na hodnotu 3 v 85 % případů a na hodnotu 10 v 5 % případů. Přiřazením postupných číselných rozsahů k jednotlivým volbám vznikne mapování 1-10 pro Look sens. = 1, 11-95 pro Look sens. = 3 a 96-100 pro Look sens. = 10. První hodnota (30) spadá do množiny 11-95, takže Look sens. je nastaven na hodnotu 3. [4]

Podobně na Obrázku 4.11 v sekci B je nastaven rozsah 1-10 pro Invert Thumbstick = Yes a 11-100 pro Invert Thumbstick = No. Druhé číslo 89 spadá do množiny 11-100, takže do sloupce Invert Thumbstick bude vložena hodnota "No“. [4]

Na Obrázku 4.11 v sekci C jsou rozsahy použití Controller Vibration pro příležitostného hráče 1-80 = Yes a 81-100 = No. Třetí náhodné číslo je 77, takže do sloupce Controller Vibration prvního testu se zadá "Yes". [4]

Sekce D na Obrázku 4.11 definuje 25 % využití pro Invert Flight Control = Yes a 75 % využití pro No. Čtvrté náhodné číslo je 25, což je v rozsahu 1-25 pro nastavení "Yes“. [4]

Poslední sekce E na Obrázku 4.11 definuje funkci Auto-Center jako 30 % = Yes a 70 % = No. Náhodné číslo pro tento test je 13, což jej řadí do rozsahu 1-30 pro nastavení "Yes". [4]

Stejným způsobem se doplní zbylé čísla a vznikne tím Tabulka 4.11. Lze si zároveň všimnout, že některé hodnoty nebyly využity vůbec (např. Look sens. s hodnotou 1) anebo se jejich četnost neshoduje s pravděpodobnostním využitím (např. Auto-Center má 30 % využití, ale objevila se ve 4 testech z 6). Důvodem může být buď nízké procento využití, nebo nízkým počtem testů pro tuto tabulku. Dalším rozdílem může být, že párové kombinatorické tabulky se sestavují vertikálně, po jednotlivých sloupcích. Tím pádem, dokud se nedokončí proces sestavování tabulky, neví se, jaké budou testovací případy ani kolik testů nakonec bude. Protože kombinatorické tabulky Cleanroom jsou konstruovány horizontálně (po řádcích), získá se tím zcela definovaný test hned na prvním řádku a na každém dalším řádku tolik kombinatorických testů Cleanroom, kolik bude chtěno. [4]

Obrácené použití Invertované použití lze použít, pokud chcete zdůraznit méně často používané funkce a chování ve hře. Vytváří se tak model použití, který by mohl odrážet způsob, jakým by hru používali lidé, kteří se snaží najít způsoby, jak hru zneužít nebo záměrně rozbít pro svůj vlastní prospěch. Pomáhá také upozornit na chyby, které unikly dřívějšímu odhalení právě proto, že se očekává, že hra bude tímto způsobem používána jen zřídka, pokud vůbec. [4]

Výpočet probíhá: [4]

1. Vypočítá se reciproční hodnota pro každou pravděpodobnost nebo pro všechny cesty z TVD.
2. Tyto reciproční hodnoty se sečtou.
3. Každá reciproční hodnota z prvního kroku se vydělí součtem všech hodnot z kroku 2.

Příklad. Existují hodnoty A = 10 %, B = 50 % a C = 40 %. V prvním kroku se vypočte pro jednotlivé údaje reciproční hodnota čísla. Pro `A: 1/0,1 = 10`, pro `B: 1/0,5 = 2` a pro `C: 1/0,4 = 2,5`. Součtem těchto vzájemných hodnot se získá součet 14,5. Touto hodnotou se vydělí hodnoty A, B a C a získají se tím převrácené hodnoty. Pro `A: 10/14,5 = 69 %`, pro `B: 2/14,5 = 13,8 %` a pro `C: 2,5/14,5 = 17,2`

%. Jednou z charakteristik tohoto postupu je, že převrací poměry mezi jednotlivými pravděpodobnostmi ve srovnání s jejich společníky pro danou sadu hodnot použití. V předchozím příkladu se B používá 5x častěji než A (50/10) a 1,25x častěji než C (50/40). Poměr mezi převráceným A a převráceným B je `69/13,8 = 5,00`. Stejně tak vztah mezi invertovaným C a invertovaným B je `17,2/13,8 = 1,25`. V případě, kdy položka má 0 %, je vhodné ke každé hodnotě před provedením tříkrokového inverzního výpočtu přičíst 0,01 %. [4]

Další příklad bude opět na HALO nastavení a využije se Obrázek 4.11. V sekci A je nastavení Look Sens. pro příležitostného hráče 10, 85 a 5. První krok: pro každou z hodnot se vypočítá reciproční hodnota. Look sens. 1: `1/0,10 = 10`. Look sens. 3: `1/0,85 = 1,176` a Look sens. 10: `1/0,05 = 20`. Druhý krok: Čísla se sečtou `10 + 1.176 + 20 = 31.176`. Třetí krok: Pro Look sens. 1: `10/31,176 = 0,32 * 100 = 32 %`, Look sens. 3: `1,18/31,176 = 0,038 * 100 = 3,8 %` a Look sens. 10: `20/31,176 = 0,642 * 100 = 64,1 %`. Porovnání invertovaných hodnot s původními hodnotami potvrzuje, že relativní podíly jednotlivých hodnot byly rovněž invertovány. Původně bylo pro Look sens. 1 použití 10 % oproti Look sens. 10, které mělo 5 %. To je poměr 2:1. V převrácené tabulce je hodnota Look sens. 10: 64,2 %, což je dvojnásobek k Look sens. 1, které má 32, 1 %. Stejným způsobem se vyplní i zbylé hodnoty (viz Tabulka 4.12). [4]

Tabulka 4.12 Hotová Invertovaná tabulka pro Look sensibility

Look Sens.	Casual	Achiever	Explorer	Multiplayer
1	32,1	99,9	60,9	75,9
3	3,8	0	8,7	5,1
10	64,1	0	30,4	19
TOTAL	100	100	100	100

Každý hráč se řídí nějakým vzorcem použití, který lze zahrnout do herních testů za účelem testování. Smyslem tohoto postupu je najít a odstranit chyby, které by se při hraní hry tímto způsobem projevily. Testy založené na obrácených profilech používání mohou zachytit vzácné chyby ve hře, které by se při milionu prodaných kopií mohly párkrát vyskytnout. [4]

4.3.4 Testovací stromy

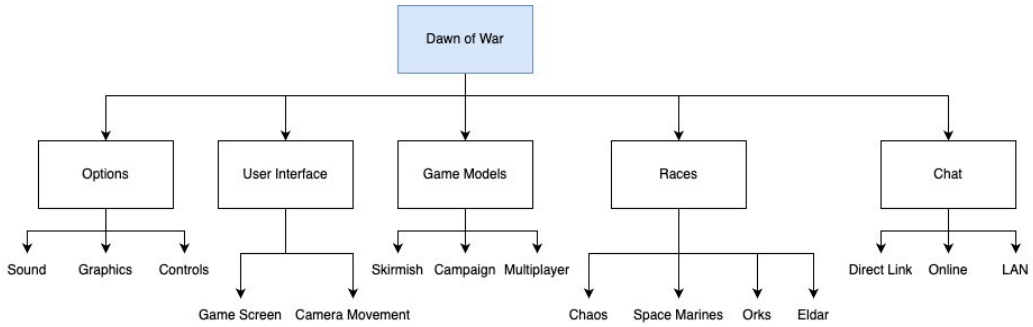
Využívají se ke třem různým účelům. Prvním jsou Stromy testovacích případů, druhým stromy testovacích prvků a třetím Návrhy testovacích stromů. [4, 32]

1. Stromy testovacích případů – dokumentují hierarchický vztah mezi testovacími případy a herními prvky, vlastnostmi a funkcemi.

2. Stromy testů nových prvků – odrážejí stromové struktury prvků nebo funkcí navržených do hry.

3. Návrhy testovacích stromů – používají se k vývoji testů, které systematicky pokrývají konkrétní herní vlastnosti, prvky nebo funkce.

Stromy testovacích případů V tomto případě již byly testy vyvinuty a zdokumentovány. Strom se použije pokaždé, když herní tým pošle testerům novou verzi. Pro příklad se použije hra Warhammer 40 000: Dawn of War (PC relatime simulace – RTS). V ní může až 8 hráčů (AI nebo opravdoví) soupeřit proti sobě. Hráči ovládají a rozvíjejí vlastní rasu válečníků, z nichž každá má své vlastní odlišné vojenské jednotky, zbraně, stavby a vozidla. Hry se vyhrávají podle různých vítězných podmínek, jako je ovládnutí lokace, obrana lokace po určitou dobu nebo úplná eliminace nepřátelských sil. [4, 32]

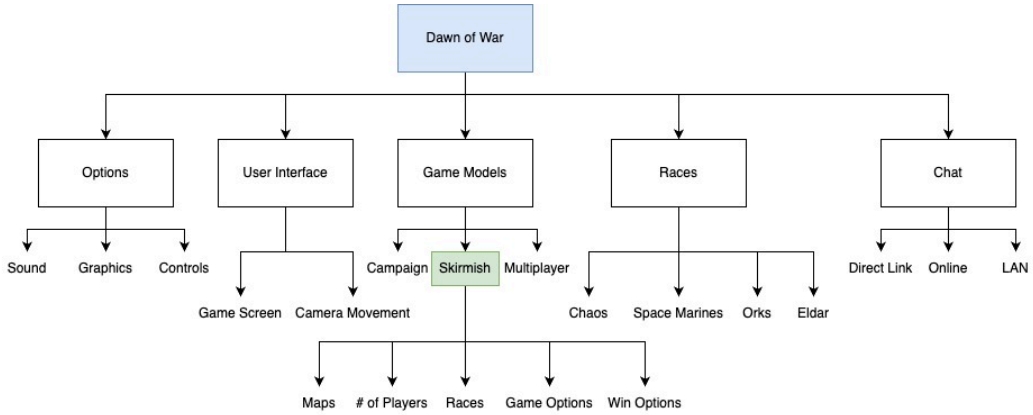


Obrázek 4.12 Strom testovacích případů I

Na vysoké úrovni lze testy Dawn of War rozdělit na testy herních možností podle Obrázku 4.12. Během vývoje hry pak může každá oprava chyby ovlivnit jednu nebo více oblastí. Díky této metodě se může tester snadno zaměřit na to, které testy spustit, a to tak, že je najdete v uzlech stromu souvisejících s částmi hry, kterých se nový kód týká. Například změna písma v editoru chatu se musí otestovat na vyšší úrovni (políčko Chat). Naopak jiné mohou být pouze pro specifické větve. Například změna způsobu, jakým je text chatu předáván online serveru. [4]

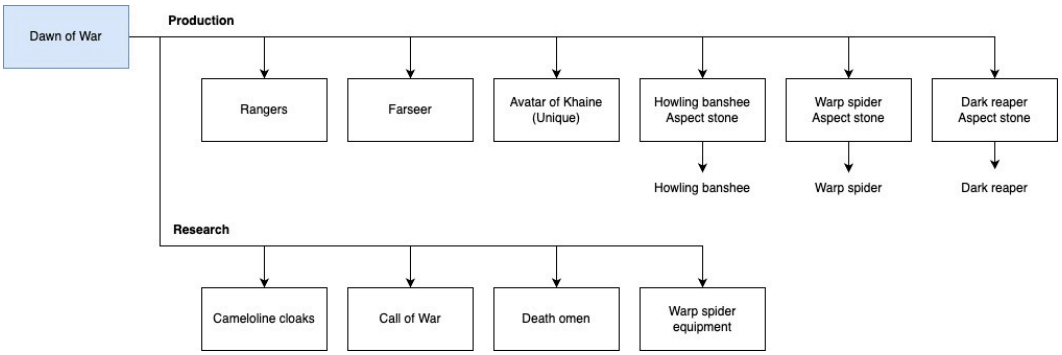
V některých případech je potřeba vybrat ještě užší výběr testů. Například testy herního režimu Skirmish by mohly být dále uspořádány podle toho, jaká mapa je použita, kolik je v zápase aktivních hráčů atp. Tento detail reprezentuje Obrázek 4.12. Odhalení dalších podrobností režimu Skirmish je důležité, protože odhaluje další sadu testů, které by měly být spuštěny, pokud dojde ke změnám herních prostředků nebo funkcí, které jsou specifické pro Rasy. [4]

Stromy testů nových prvků Další využití se používá k zobrazení skutečných stromových struktur funkcí, které jsou implementovány ve hře. Hra Dawn of War má



Obrázek 4.13 Strom testovacích případů II

takové struktury pro technologické stromy jednotlivých ras. Tyto stromy definují pravidla závislosti, podle kterých lze generovat jednotky, vozidla, struktury a schopnosti. Například, než může Eldar vyrobit jednotky Howling Banshee, musí nejprve postavit Aspektový portál a vylepšit strukturu pomocí Aspektového kamene Howling Banshee. Ale jiné jednotky (například Stráže) lze vyrábět okamžitě. Tyto stromy mohou být poměrně složité, se závislostmi mezi více stavbami, vylepšeními a výzkumnými předměty. Obrázek 4.14 ukazuje technologický strom portálu aspektu pro rasu Eldar. [4, 32]



Obrázek 4.14 Technologický strom pro Eldar Aspect Portal

Jako další příklad lze považovat Final Fantasy Tactics Advance (FFTA) a jejich stromy povolání definované pro jednotlivé rasy postav. Postavy si musí osvojit určitý počet dovedností v jednom nebo více povoláních, než se jim zpřístupní nové volby povolání a jim odpovídající dovednosti. Měla by existovat sada testů pro tento strom, která postupně vynechá každou předběžnou podmínku a navíc test, při kterém jsou všechny splněny. V každém kroku by se mělo kontroval nové nastavení a chování, a to i předčasně, kdy by ještě nastavení nemělo být dostupné. [4]

Pracovních cesty ve FFTA jsou většinou jednoduché, kdy povolání modrého mága a iluzionisty jsou dostupná až poté, co se herní postava naučí schopnosti dvou dalších povolání – bílého a černého mága. Na Obrázku 4.15 jsou zobrazeny stromy povolání. [4]

Obrázek 4.15 Stromy povolání modrého mága a iluzionisty pro lidi ve hře FFTA

Pro příklad se definují testy pro konkrétní úlohu zadáním testovacích hodnot pro jednotlivé uzly podél větví stromu. Obrázky 4.12 a 4.13 ukazují testovací případy, které by se měly získat pro stromy na Obrázku 4.15. [4]

Tabulka 4.13 Testy vlastností stromu pro lidského modrého mága

White Mage abilities	Black Mage abil.	Blue Mage abil.
0	1	No
1	0	No
1	1	Yes

Jednotlivé sloupce reprezentují schopnosti bílého, černého a modrého mága, zatímco řádky označují kombinace těchto schopností. Každá buňka obsahuje informaci o tom, zda je daná kombinace schopností vhodná pro modrého mága (označeno "Yes") nebo ne (označeno "No"). [4]

Tabulka 4.14 Testy vlastností stromu pro lidského iluzionistu

White Mage abilities	Black Mage abil.	Blue Mage abil.
2	5	No
3	4	No
3	5	Yes

Existuje mnoho dalších stromů, které se ve hrách mohou vyskytovat. Například Odemykání kódů, vylepšení nebo vylepšení; Odemykání nových map, prostředí nebo závodních tratí; Postup v turnajích; Struktury nabídek herních možností. Je důležité testování situací, kdy různé stromy nabídek nebo cesty stromů mohou ovlivňovat stejnou hodnotu, schopnost nebo herní prvek. Například herní možnosti Dawn of War nastavené v režimu Skirmish se stávají také hodnotami používanými v režimu Multiplayer, LAN, Online nebo Direct Host. Tyto hodnoty by se měly nastavovat a kontrolovat všemi možnými prostředky (cestami), které hra poskytuje. [4]

Návrhy testovacích stromů V případech, kdy je hra dost komplexní, mohou některé prvky působit chaoticky. Stejně, jako v případě Stromů testů nových prvků slouží k

určení úzkého rozsahu testů, které se mají spustit po změně v kódu (tento strom testů může určit úzký rozsah chování hry, které je třeba opravit, když je nalezena chyba). Vždy se musí brát v potaz budoucí aktualizace. Pokud se vytvoří strom (například ras stvoření, které mají mezi sebou různá pravidla a hráč si může mezi nimi vybírat) který se aktualizuje (např. o novou rasu), je potřeba se vrátit ke kroku přiřazení k jednotlivým větvím a použít je jako výchozí bod pro nové dokončení. [4, 32]

Tato technika návrhu je také užitečná pro vyplnění "prázdných míst"jakékoli funkce, kde již máte zkušenosti nebo intuici o tom, co by se mělo testovat, ale chcete zajistit úplnější testování, zejména pokud jde o různé interakce nebo situační možnosti. Použití této techniky může být použito i pro návrhy testovacích stromů k počítačové umělé inteligenci soupeře, síťovému zpracování herních zpráv nebo interakce a kombinace kouzel. [4, 32]

4.3.5 Play a Ad Hoc testování

Tato kapitola se soustředí na více chaotický, nestrukturovaný, ale stejně důležitý přístup. Ad hoc Testování je označován také jako "obecné testování"a popisuje hledání defektů v méně strukturovaném způsobu. Play testing naopak spočívá ve hraní hry za účelem testování subjektivních vlastností jako je vyváženost, obtížnost a často těžko definovatelný "zážitek ze hry". [4, 33]

Ad Hoc testování I přes pečlivě prozkoumání, vždy existuje možnost, že se něco přehlédnete. Toto testování umožňuje jednotlivému testerovi prozkoumat investigativní cesty, které se mu mohly objevit, i když podvědomě, během provádění strukturovaných testovacích sad. Existují dva hlavní typy ad hoc testování. První je volné testování, které profesionálnímu hernímu testerovi umožňuje "odejít z předepsaného scénáře"a improvizovat testy na místě. Druhým typem je řízené testování, které má za úkol vyřešit konkrétní problém nebo najít specifické řešení. [4, 33]

Ad hoc testování umožňuje testerům se podívat na hru s "čerstvýma očima". Pomáhá zkoumat různé módy a funkce hry, které leží mimo jejich hlavní oblast zodpovědnosti. To může zahrnovat přiřazení členů týmu multiplayeru ke hraní kampaně pro jednoho hráče nebo poslání testerů z jiného projektu strávit den (nebo část dne) s testováním dané hry. Tento přístup umožňuje objevit přehlédnuté problémy a nedostatky, které by jinak mohly zůstat nepovšimnuty. [4, 33]

Je důležité si stanovit cíle, kterých by se měl tým nebo jednotlivec držet. Cíl by měl být jednoduchý, ale výstižný (např. *Jak daleko se můžu dostat v příběhu hry pokud budu odpovídat v dialozích pouze negativně?*). Vždy by se mělo testování nahrávat nebo nějak zaznamenávat a nemělo by se myslet ve skupinách. Jedním z běžných aspektů

skupinového myšlení je sklon k autocenzuře – kdy jednotlivci ve skupině nevyjadřují pochybnosti nebo nesouhlas ze strachu, že budou kritizováni, ostrakizováni nebo ještě hůře. Tester se tedy musí vyhnout větám typu *Všichni hráli StarCraft, takže nemusíme testovat tutoriál v naší vlastní RTS.* nebo *Nikdo nepoužívá zbraně pro boj zblízka, takže budu používat jen pistole.* [4, 33]

Řízené testování lze nejlépe popsat jako "detektivní testování", protože má specifický, vyšetřovací charakter. Nejjednodušší forma cíleného testování odpovídá na velmi konkrétní otázky, jako *Funguje nová kompilace?*, *Dají se cut-scény přerušit?* nebo *Je ukládání stále nefunkční?*. Testeři musí umět v případě řízeného testování podat informaci o počtu reprodukce chyby. Informaci o reprodukovatelnosti chyb v hlášeních o chybách se obvykle vyjadřuje jako procento nebo jako počet pokusů, které vedly k pozorování chyby. Tato informace je důležitá pro testování her, zejména v případě, kdy se chyba týká statických prvků, jako jsou chyby v textu hry. Je-li chyba reprodukovatelná, je pravděpodobnější, že bude opravena. [4, 33]

Při testování by se mělo postupovat jako při vědeckém výzkumu. Stejně jako výzkum totiž i testování se snaží pozorovat nějaký jev, vytvořit teorii o tom, co jej způsobilo a potvrdit tuto teorii. V případě chyby (například pádu hry) se postupuje následovně: [4]

1. Rychle se poznamenají všechny informace o tom, co tester dělal, když k závadě došlo. Prohlédne si záznam a určí poslední věc, kterou ve hře dělal před jejím pádem.

2. Zpracujte všechny tyto informace a co nejlépe odhadne, jaká konkrétní kombinace a pořadí vstupů mohla pád způsobit.

3. Projde jednotlivé kroky cesty, dokud s nimi nebudete spokojen. Odhadne, že pokud je zopakujete, závada se objeví znovu.

4. Restartujte počítač, znovu spustí hru a opět proveďte definované kroky.

5. Pokud se výsledek podařil, zapíše si jej. V opačném případě změní jeden (a pouze jeden) krok ve své cestě a zkusí to znova. Takto pokračuje, dokud na problém znova nenarazí.

Play testování Předešlé kapitoly byly spíše o testování hry, kdy se pokládala otázka *Funguje tato hra?*. Ovšem Play testování se zaměřuje na možná ještě důležitější otázku *Funguje tato hra dobře?*. V této otázce je možné zahrnout další otázky typu *Je hra moc snadná?*, *Je ovládání intuitivní?*, *Je tato hra zábavná?*, atp. Zákon o rovnosti zde hraje důležitou roli, kdy se musí tester ujistit, že je hra například dlouhá, ale ne moc

dlouhá, jednoduchá na naučení, ale není zjednodušená nebo je dostatečnou výzvou, ale není frustrující. [4, 33]

Jakmile je hra připravena k hernímu testování, je důležité, aby zpětná vazba z testů byla stejně konkrétní a prezentovaná stejně organizovaně a podrobně jako jakákoli jiná zpráva o závadě. Například věta: [4]

Lotus Warlocks způsobují příliš velké poškození a je třeba jim ubrat.

Je neadekvátní, protože tester není konkrétní. Jak velké poškození je příliš velké? V poměru k čemu? Pokud „jim ubrat“ znamená "méně silný", o kolik méně? 50 %? O 50 bodů? Vývojový tým tuto připomínku pravděpodobně nebude brát vážně, protože si myslí, že jde o impulzivní, emotivní reakci. [4]

Lotosoví čarodějové jsou silnější než jednotky nejvyšší úrovně ostatních tří ras. Jejich útok způsobuje přibližně o 10 % větší poškození než útok dračího samuraje, hadího ronina a vlkodlaka z klanu vlků. Za stejnou dobu, jakou potřebují těžké jednotky ostatních klanů na dva útoky, provedou tři útoky. Hráči, kteří se rozhodnou hrát za Lotosový klan, vyhrávají 75 % svých her. [4]

Na druhou stranu tento komentář je konkrétní a založený na faktech. Poskytuje výrobcům a designérům dostatek informací, aby mohli začít přemýšlet o změně vyvážení jednotek. Nenavrhuje však, jak by se měl problém řešit. Pro správné navrhnutí řešení je dobré si u sebe potvrdit, že je to opravdu dobrý nápad. Následně si tento nápad nechat trochu rozležet v hlavě a prodiskutovat jej s kolegy. Pokud se na tom vedoucí shodne, nápad bude realizován. [4]

S postupem vývoje se stává obtížnost hry obtížněji definovatelnou, protože tým vývojářů a testovatelů se stává příliš obeznámený s hrou a ztrácí schopnost objektivně posoudit její obtížnost. Tým se stává nejlepšími hráči vlastní hry na světě, což má své výhody, ale také své nevýhody. S přibývajícím časem se ztrácí schopnost vnímat hru jako novou a čerstvou, což vede k subjektivnímu hodnocení obtížnosti. Je proto nezbytné získat nový pohled na hru od externích hráčů. Externí testování začíná mimo testovací a vývojové týmy, ale stále uvnitř společnosti. Je dobré, aby si každý, kdo je ochoten, od CFO po částečně zaměstnanou recepční, vyzkoušel hru, pokud zůstávají nezodpovězené otázky. Zde musí být opatrní a pamatovat na varování Dr. Wernera Heisenberga, že samotný akt pozorování něco mění na pozorované skutečnosti. [4, 33]

Testování v oboru (Subject Matter Testing) zahrnuje konzultaci hry s odborníky na dané téma, pokud je hra postavena na základech z reálného světa. Například během vývoje simulátoru PC stíhačky Flanker producenti ve vydavatelství SSI komunikovali s malou skupinou amerických a ruských stíhacích pilotů, kteří obdrželi beta verzi hry. Jejich zpětná vazba ohledně realismu hry, od pocitu letadel po nápisy v ruštině na přístrojových štítcích, se ukázala jako nepostradatelná a recenze po vydání byly díky

tomu velmi pozitivní. [4, 33]

Beta testování může poskytnout velmi užitečná data, ale pokud není řízeno správně, může poskytnout spoustu zbytečných informací. Existují dva typy Beta testování: uzavřené a otevřené. Uzavřené Beta testování probíhá jako první a je pečlivě kontrolováno. Beta testery jsou pečlivě vybíráni a obvykle musí odpovědět na mnoho otázek, než budou přijati do testu. Nejjednodušší typ uzavřeného Beta testování probíhá na konzolích nebo jiných offline platformách. Testeři jsou naverbováni, aby přišli do kanceláří vydavatele nebo vývojářů, zkontrolovali a hráli hru a poté vyplnili dotazník nebo se účastnili diskuse ve skupině. Otevřené Beta testování probíhá po uzavřeném Beta testování. Otevřené Beta testování je otevřeno všem, kteří mají zájem o účast. I když vývojáři stále požádají tuto mnohem větší skupinu o zpětnou vazbu ohledně hratelnosti, jejich role může být především v testování síťového kódu a vyhodnocení přihlašovacího systému, vytváření skupin, celkové stability sítě, zpoždění atd. Beta testery neprovádějí testovací scénáře, ale hlásí chyby a poskytují zpětnou vazbu ohledně hratelnosti. Ti pak ve formě formuláře vyplňují hlášení chyb, které obdrží manažeři nebo QA oddělení. [4, 33]

Kapitola byla zaměřena na Ad hoc testování, které umožní prozkoumat hru a procházet jí jako bludištěm. Existují dva hlavní typy Ad hoc testování. Prvním je volné testování, které profesionálnímu testerovi her umožňuje "odchýlit se od scénáře"a improvizovat testy za běhu. Druhým je cílené testování, jehož cílem je vyřešit konkrétní problém nebo najít konkrétní řešení. [4, 33]

II. PRAKTICKÁ ČÁST

5 ANALÝZA TESTOVACÍHO PROCESU: POHLED Z PRAXE

Kapitola se zaměří na seznam otázek, které byly položeny herním studiím v rámci této diplomové práce. Cílem bylo získat hlubší pochopení testovacího procesu a perspektivu týmu QA v reálném herním studiu. Tento přístup umožnil získat přehled o praktických zkušenostech a postřezích z reálného prostředí herního vývoje. Odpovědi budou rozebrány a analyzovány.

5.1 NOXGames

Jedná se o Brněnské indie studio, které vzniklo v roce 2005. Studio vydává mobilní hry, kdy k aktuálnímu dni jich podle jejich webu vydali již přes 150. Vytvořili RPG, strategie, karetní nebo hypercasual hry. Hry vydávají na platformy Google a Apple. Mezi jejich hry patři například Assassin Hero: Infinity Blade, Shadow Deck: Magic Heroes CCG, War Hex: 4X Válečná Strategie nebo Merge Neon Car: Idle Car Merge.

5.1.1 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)? Postupy 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 využíváte? 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 způsoben 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.

5.1.2 Souhrn

Studio organizuje testování prioritně podle releasů, využívá manuální i okrajově automatické testování a vývoj softwaru provádí podle spirálového modelu. Nejčastěji používanými testy jsou akceptační, integrační a systémové testy. Regresní testování je zahájeno již při programátorské práci. Multiplayer hry nejsou v jejich portfoliu, ale testování na různých platformách (Android/iOS) je důkladné. Bezpečnost a stabilitu zajišťují pečlivým testováním a spoluprací s vývojáři. Automatizované testování využívají okrajově s Unity Test Framework. Pro správu a dokumentaci testovacích procesů používají ClickUp, Miro a Google SpreadSheet. Prioritními oblastmi testování jsou funkčnost a výkon aplikace. Testování fyziky, map, textur a objektů probíhá prostřednictvím testovacích scénářů a konzultací s vývojáři. Celkově klade studio důraz na kvalitu produktů prostřednictvím systematického testování a spolupráce s ostatními týmy v procesu vývoje softwaru.

5.2 SCS Software s.r.o.

SCS Software s.r.o. je české herní studio se sídlem v Praze, které bylo založeno v roce 1997. Soustředí se primárně na tvorbu desktopových a konzolových her, které jsou simulátory. Mezi jejich nejznámější hry patří Euro Truck Simulator 2, American Truck Simulator a World of Trucks.

5.2.1 Dotazník

Jak organizujete testování ve vašem herním studiu? QA team máme rozdělené na sub-teamy, 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? Povětšinou přímé manuální testování. Máme však i automatické validační testy.

Jaký máte proces vývoje softwaru? A jak často podle něj testujete? Kontrola kvality začíná již v preprodukci. 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 mody 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 způsoben 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.

5.2.2 Souhrn

Toto české herní studio se specializuje na vývoj simulátorů pro desktopové a konzolové platformy jako Euro Truck Simulator 2 a American Truck Simulator. Testování ve studiu je organizováno pomocí QA týmu rozděleného do sub-týmů pro různé aspekty testování, kde kombinující manuální a automatické testování. Kontrola kvality začíná již v preprodukci a pokračuje během celého vývoje, s důrazem na stabilitu produktu a kvalitní uživatelský zážitek. Pro správu testovacích procesů využívají nástroje jako Favro a Mantis a jejich nejvyšší prioritou při testování je stabilita produktu, následovaná kvalitním uživatelským zážitkem. Manuální testování se provádí napříč různými aspekty hry s pomocí iterativního přístupu a zapojením různých testerů, a to včetně fyziky, map, textur a objektů.

6 TESTOVACÍ METODIKY

V této kapitole se přiblíží metodiky z knihy [4] a budou aplikovány buď na hru, která je ve vývoji nebo na již existující hry. Budou obsahovat Kombinatorické testování, Testování vývojové diagramy a jejich kombinaci dohromady.

6.1 Kombinatorické testování

V rámci vytvoření kombinatorického testu se využije nastavení interface ze hry Sid Meier's Civilization V. Jedná se o heterogenní test, který má za cíl otestovat co nejpodrobněji možnosti nastavení interface. Z celkového počtu kombinací $3^4 2^9$ (41 472 kombinací) vzniklo 9 testů, které pokrývají většinu možností.

Obrázek 6.1 Civilization V Interface menu

Nastavení obsahuje celkem 14 položek, kdy devět z nich je typu bolean a obsahuje jednoduchý checkbox s možností TRUE/FALSE. Následují dvě položky, které jsou typu integer a dávají možnost zadat celá čísla v rozmezí 0 až 999. Dále dvě položky, které jsou opět typu integer a nabízí za pomocí slideru vybrat hodnotu v rozmezí 0 až 2 s krokem 0,1. A na závěr select box s výběrem možnosti jazyka, který nabízí jedinou možnost, a to Angličtinu. Níže seznam hodnot s jejich typem:

- Alternate Cursor Zoom Mode - true/false
- Show All Policy Information - true/false
- Auto Unit Cycle - true/false

- Single Player Score List - true/false
- Multiplayer Score List - true/false
- Enable Map Inertia - true/false
- Skip Intro Video - true/false
- Automatically Size Interface - true/false
- Use Small Scale Interface - true/false
- Map Drag Speed
  - min: 0, max: 2
  - krok 0,1
  - default: 1
- Map Zoom Speed
  - min: 0, max: 2
  - krok 0,1
  - default: 1
- Spoken Language - select box s hodnotami; default: English
- Turns Between Autosave
  - min: 0, max: 999
  - krok 1
  - default: 10
- Max Autosaves Kept
  - min: 0, max: 999
  - krok 1
  - default: 5

Pro test a vytvoření kombinatorické tabulky se použijí u integer polí hodnoty: Map Drag Speed: 0 - 0,9 - 1 Map Zoom Speed: 0 - 1,1 - 1 Turns Between Autosave: 0 - 10 - 999 Max Autosaves Kept: 0 - 5 - 999

Tabulka bude kvůli délce rozdělena do čtyř menších tabulek. Jeden řádek se rovná jeden testovací scénář, který se použije. První se začne těmi položkami, které mají

největší výběr. V tomto případě to jsou 4 položky, které mají na výběr ze 3 možností. Pro první sloupec *Map Drag Speed* se vypíší hodnoty za sebou. Druhý sloupec *Map Zoom Speed* se vypíše v pořadí, aby vznikla vždy dvojice, která je ojedinělá. Třetí sloupec *Turns Between Autosave* proběhne stejným způsobem. Pro čtvrtý sloupec *Max Autosaves Kept* je již nemožné udělat shodu se všemi předchozími sloupci. Vždy bude shoda pouze 2 ze 3. Viz Tabulka 6.1.

Tabulka 6.1 Kombinatorická tabulka pro Interface settings v Civilization V (část I.)

Map Drag Speed	Map Zoom Speed	Turns Between Autosave	Max Autosaves Kept
0	0	0	0
0.9	1.1	10	5
1	1	999	999
0	1.1	999	0
0.9	1	0	5
1	0	10	999
0	1	10	0
0.9	0	999	5
1	1.1	0	999

Následuje Tabulka 6.2 a její čtvrtý sloupec *Alternate Cursor Zoom*. První až třetí řádek je hodnota *FALSE*. Tímto se pro všechny předchozí hodnoty zaručí, že budou mít shodu 4 ze 4. Čtvrtý až šestý řádek je hodnota *TRUE*. Opět se tím zaručí, že hodnoty budou mít shodu s ostatními 4 ze 4. Zbylá řádky již nedokážou vytvořit novou shodu, tudíž je náhodně zapíšeme podle 7. pravidla.

Pro sloupec *Show All Policy Information* se v případě prvních dvou řádků nastaví *TRUE* a *FALSE*. Pro třetí a čtvrtý řádek se nastaví *FALSE* a *TRUE*, které mají možnost vytvoření nových dvojic ze 4 z 5 případů. Pátý a šestý řádek s hodnotou *TRUE* bude mít 3 z 5 případů novou dvojici (narozdíl od *FALSE*, které by mělo pouze 2 z 5). V případě nastavení *TRUE* u sedmého řádku by nastalo 0 nových dvojic. Nastaví se tedy *FALSE* se shodou 2 z 5. Osmý řádek bude míst shodu 1 z 5 v případě nastavení *FALSE*. Poslední řádek stejně jako ten výše bude mít *FALSE* se shodou 2 z 5.

Další sloupec je *Auto Unit Cycle* kdy první dvě hodnoty jsou *TRUE* a *FALSE*. Pro další řádek vyjde na shoda na 4 z 5 u obou variant, takže se zvolí *FALSE*. U čtvrtého řádku vyjde lépe *FALSE* a u pátého *TRUE* se shodou 3 z 5. Šestý řádek se shodou 3 z 5 je výhodnější pro *TRUE*. Sedmý, osmý a devátý je nastaven na *FALSE*. V opačném případě měla hodnoty *TRUE* 0 z 5. Pokud se vyfiltruje hodnota *TRUE*, tak *Multiplayer Score List* bude mít pouze hodnoty *FALSE* a *Use Small Scale Interface* bude mít pouze hodnoty *TRUE*. Zároveň *Turns Between Autosave* neobsahuje hodnotu 0 a *Map Zoom Speed* neobsahuje hodnotu 1.

Tabulka 6.2 Kombinatorická tabulka pro Interface settings v Civilization V (část II.)

Alternate Cursor Zoom Mode	Show All Policy Information	Auto Unit Cycle
TRUE	TRUE	FALSE
FALSE	FALSE	TRUE
FALSE	FALSE	FALSE
TRUE	TRUE	TRUE
TRUE	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	FALSE	FALSE
FALSE	FALSE	FALSE
FALSE	FALSE	FALSE

Třetí část Tabulky 6.3 obsahuje sloupce *Single Player Score List*, *Multiplayer Score List* a *Enable Map Inertia*. Pro sloupec *Single Player Score List* jsou první dvě hodnoty *FALSE* a *TRUE*. Třetí řádek má pro obě hodnoty shodu 5 ze 7, tudíž se vybralo náhodně *TRUE*. Čtvrtý a pátý řádek má hodnoty *FALSE* se shodou 4 ze 7 a 3 ze 7. Pro šestý řádek byla opět shoda 3 ze 7 pro obě hodnoty, takže se vybrala *TRUE*. Sedmý a devátý řádek měl shodu 2 ze 7 pro *FALSE*. Nakonec osmý řádek měl shodu 0 ze 7 pro obě hodnoty. Byla tedy náhodně vybrána hodnota *TRUE*. Pro tento sloupec v případě nastavení filtru na *TRUE* se u sloupců *Map Drag Speed*, *Turns Between Autosave* a *Max Autosaves Kept* neeviduje nikde hodnota 0.

Prostředním sloupcem tabulky je *Multiplayer Score List*. Ten pro první řádek zvolil hodnotu *TRUE* a pro druhý *FALSE*. Třetí řádek se shodou 6 z 8 byl nastaven na *TRUE* a čtvrtý se stejnou shodou na *FALSE*. Pro pátý řádek byla stejná shoda u obou variant. Vybrala se tedy náhodně hodnota *TRUE*. Šestý a sedmý řádek mají oba hodnotu *FALSE* a osmý a devátý zase *TRUE*. U osmého řádku byla pro obě varianty shoda 0 z 8 a pro hodnotu obecně pro celý sloupec u hodnoty *TRUE* je vždy nastavena hodnota *FALSE* pro sloupec *Auto Unit Cycle*. A pro sloupec *Turns Between Autosave* chybí hodnota 10.

Poslední sloupec je *Enable Map Inertia* s prvními hodnotami *TRUE* a *FALSE*. Pro třetí a čtvrtý řádek vyšly nastejno obě hodnoty, ale vybrána byla *FALSE*. Dále byla nastavena hodnota *TRUE* pro pátý, šestý a sedmý řádek, kdy všude vyšla výhodněji (nejméně v sedmém řádku se shodou 1 z 9). U osmého a devátého řádku vyšlo nastejno, jaká bude hodnota (u obou případů 1 z 9). Vybrána byla tedy *TRUE*. Pokud se vyfiltruje hodnota *TRUE*, tak ve sloupci *Turns Between Autosave* chybí hodnota 999.

Pro Tabulku 6.4 se vyberou poslední tři nastavení (*Skip Intro Video*, *Automatically Size Interface* a *Use Small Scale Interface*). U prvního sloupce se jako první vybralo *FALSE* a *TRUE*. Třetí řádek bude mít *FALSE* se shodou 7 z 10. Pro čtvrtý řádek

Tabulka 6.3 Kombinatorická tabulka pro Interface settings v Civilization V (část III.)

Single Player Score List	Multiplayer Score List	Enable Map Inertia
FALSE	TRUE	TRUE
TRUE	FALSE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	FALSE	TRUE
TRUE	TRUE	FALSE
FALSE	TRUE	FALSE

vyšla lépe shoda 7 z 10 u *TRUE* oproti *FALSE* se shodou 4 z 10. U pátého řádku zase bude hodnota *FALSE*. *FALSE* je nastaveno pro šestého řádku se shodou 4 z 10. Na druhou stranu hodnota *TRUE*, i když pro oba atributy vyšla shoda 0 z 10. Osmý řádek má hodnotu *FALSE* a devátý uzavírá první sloupec s hodnotou *TRUE*.

Pro prostřední sloupeček s názvem *Automatically Size Interface* se začalo s hodnotami *TRUE* a *FALSE*. Třetí řádek se shodou 7 z 11 byl nastaven na *FALSE*. Pro čtvrtý řádek opět se shodou 7 z 11 bylo nastaveno *TRUE* a *FALSE* pro řádek pátý. U šestého a sedmého řádku se nastavilo *TRUE* (shoda 4 a 2 z 11). A poslední dva řádky byly nastaveny na *FALSE*. Poslední řádek měl 0 z 11 u obou variant. Hodnota tedy byla vybrána náhodně.

Tabulka 6.4 Kombinatorická tabulka pro Interface settings v Civilization V (část IV.)

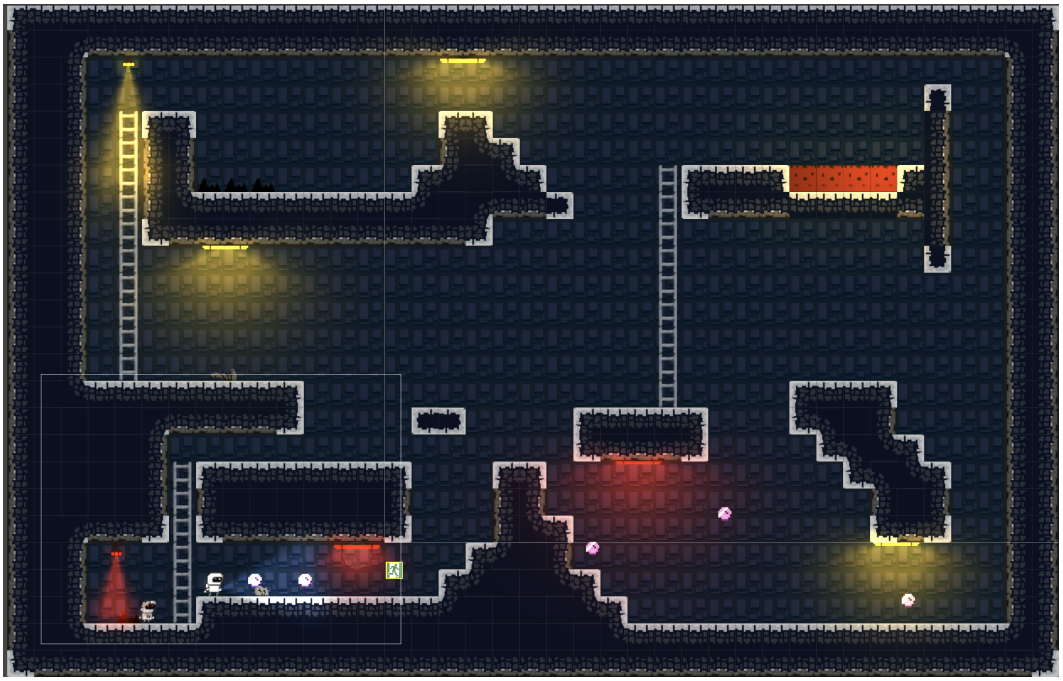
Skip Intro Video	Automatically Size Interface	Use Small Scale Interface
FALSE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	FALSE	FALSE
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE
TRUE	FALSE	FALSE

Pro poslední sloupec Tabulky 6.4 s názvem *Use Small Scale Interface* se jako první vybere hodnota *FALSE* a poté *TRUE*. O jeden bod vyšší shodou se vybrala na třetí řádek hodnota *FALSE*, která měla 8 z 12. Následující tři řádky se nastavily na *TRUE*, kdy jejich shoda byla 7 z 12 a 4 nebo 5 z 12. U sedmého řádku byla u možnosti *TRUE* shoda 0 z 12, takže se vybrala hodnota *FALSE* se 3 novými dvojicemi. Pro osmý řádek

byl rozdíl jedné dvojice o kterou se vybralo opět *FALSE* (shoda 2 z 12). A pro poslední řádek se v případě jediné nové dvojice propsala hodnota *FALSE*. Pokud se vyberou u tohoto sloupce pouze hodnoty *FALSE*, tak pro sloupec *Auto unit cycle* nebude nikdy nastavena možnost *TRUE*.

6.2 Testovací vývojové diagramy

Kapitola bude obsahovat diagramy pro testovací level ve vyvíjené hře. Probere se chování v levelu na základě možností, které hráč má (pohyb, smrt a sběr mincí). První se vytvoří samotný diagram, poté jeho datový slovník a na závěr jsou sepsány cesty diagramu.



Obrázek 6.2 Ukázka vyvíjené hry ve světle

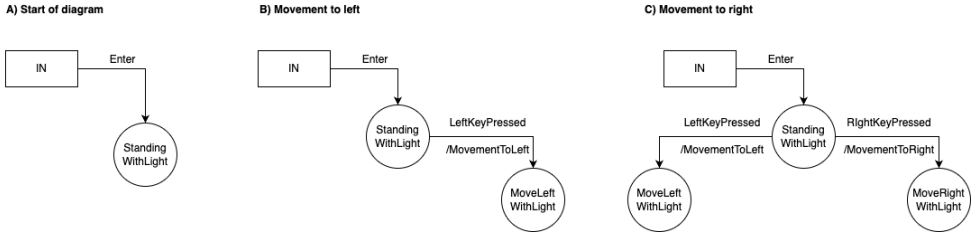
6.2.1 Vytvoření vývojového diagramu

Diagram bude vycházet ze screenshotu vyvíjené hry (viz Obrázek 6.2) ve kterém se popíší možnosti hráče v daném levelu. Pro lepší přehlednost na obrázku bylo ve hře zvýšeno osvětlení. Ve standartním stavu je tma a hráč vidí jen přes svou baterku nebo osvětlení.

Hra začne a hráči se zobrazí úroveň spolu s kosmonautem, který stojí a svítí si baterkou. Z "IN"obdélníku se vytvoří flow do stavu *StandingWithLight*. Název události je "Enter"(ID se uvede až na konci). Tato část je viditelná na Obrázku 6.3 v části A.

Následuje vytvoření první reakce, kterou je pohyb doleva. Vytvoří se nový stav *MoveLeftWithLight*, ke kterému se napojí flow *LeftKeyPressed* s akcí *MovementToLeft*.

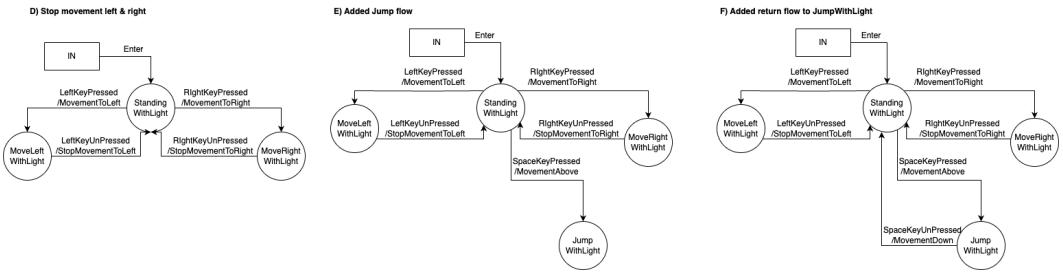
Součástí akce je jak pohyb daným směrem, tak i změna osvětlení dané strany. Tento progres je představen na Obrázku 6.3 v části B. Pokračuje se vytvořením druhé reakce se stavem *MoveRightWithLight*, ke kterému je napojené flow *RightKeyPressed* s akcí *MovementToRight*. Tento proces je zaznamenám v části C Obrázku 6.3.



Obrázek 6.3 První část vývojového diagramu pro pohyb

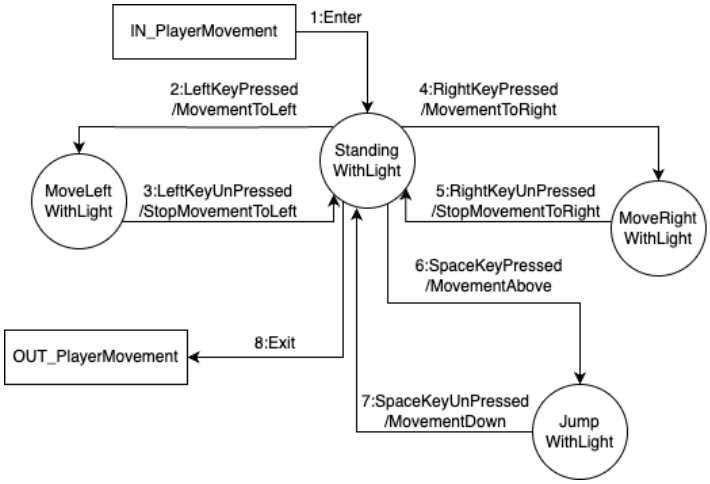
Pro Obrázek 6.4 část D se vytvořily zpáteční flows pro stavy *MoveRightWithLight* a *MoveLeftWithLight*. Ke stavu *MoveLeftWithLight* se vytvoří flow *LeftKeyUnPressed* s akcí *StopMovementToLeft*. Reprezentuje to okamžik, kdy hráč pustí klávesu pro pohyb doleva a kosmonaut se zastaví. Obě nové flows směřují zpět do stavu *StandingWithLight*. Ke druhému stavu *MoveRightWithLight* se vytvoří flow *RIghtKeyUnPressed* s akcí *StopMovementToRight*. To reprezentuje situaci, kdy hráč pustí klávesu pro pohyb doprava a kosmonaut se zastaví.

Pro část E se přidal nový stav reprezentující skok hráče s názvem *JumpWithLight*. Ten má flow *SpaceKeyPressed* s akcí *MovementAbove*. V případě kliknutí klávesy Space (mezerník), kosmonaut vyskočí. Akce je jednorázová a po kliknutí a vyskočení začne kosmonaut zase padat. To je ukázáno na části F, kdy přibylo nové flow *SpaceKeyUnPressed* s akcí *MovementDown*. Flow směřuje zpět do stavu *StandingWithLight*.



Obrázek 6.4 Druhá část vývojového diagramu pro pohyb

Nakonec se pro diagram přidá čtverec "OUT"a vytvoří se událost *EXIT*. Pro jednotlivé flows se přidají ID a do obdélníků "IN"a "OUT"se doplní název diagramu. Kompletní vývojový diagram je k vidění na Obrázku 6.5.



Obrázek 6.5 Finální vývojový diagramu pro pohyb

6.2.2 Vytvoření datového slovníku

Z diagramu na Obrázku 6.5 se vytvoří Datový slovník. Ten obsahuje seznam všech stavů a událostí, které se zapisují jako odrážky pro lepší kontrolu a vizuální rozdělení. Seznam je napsán formou checkboxů pro lepší kontrolu, jak bylo zmíněno v teoretické části práce. Datový slovník bude obsahovat pouze 6 stavů, protože události a stavy jsou v tomto případě dost podobné.

`StandingWithLight`

☐ Animace pro stání je aktivní.

☐ Světlo svítilny svítí.

☐ Postava se nehýbe.

`MoveLeftWithLight`

☐ Animace pro pohyb vlevo je aktivní.

☐ Světlo svítilny svítí daným směrem.

☐ Postava se hýbě daným směrem.

`MoveRightWithLight`

☐ Animace pro pohyb vpravo je aktivní.

☐ Světlo svítilny svítí daným směrem.

☐ Postava se hýbe daným směrem.

`JumpWithLight`

☐ Světlo svítilny svítí daným směrem.

☐ Postava se hýbe daným směrem.

`IN_PlayerMovement`

☐ Spuštění dané úrovně a zpřístupnění pohybu hráči.

`OUT_PlayerMovement`

☐ Ukončení dané úrovně smrtí, vítězstvím nebo ukončením úrovně.

6.2.3 Vytvoření cest

Na závěr se vytvoří cesty, které jsou řada flows, které se prochází ve specifickém pořadí a vytváří tímto testovací scénáře. Podle cest se pak provádí testy buď manuální nebo automatizované.

Začne se tedy od IN a přejde se z flow 1 do *StandingWithLight*. Z tohoto stavu se přejde přes flow 2 do *MoveLeftWithLight* a pokračuje se přes flow 3 opět do *StandingWithLight*. Aktuálně je tedy cesta **1,2,3**. Následuje flow 4, přes které se dostane *MoveRightWithLight* a od něj se pokračuje díky flow 5 zpět do *StandingWithLight*. Aktuálně je tedy cesta **1,2,3,4,5**. V poslední fázi se vyzkouší skok za pomoci flow 6 a flow 7. A přes flow 8 se cesta ukončí. Výsledná cesta je **1,2,3,4,5,6,7,8**. Může se jednat o minimální cestu, která pokrývá veškerá flows v diagramu. Klidně může být i Výsledná cesta **1,4,5,2,3,6,7,8** nebo **1,6,7,2,3,4,5,8**.

Vytvoří se testovací scénář z cesty **1,6,7,2,3,4,5,8**.

`Spuštění dané úrovně a zpřístupnění pohybu hráči`

☐ Zkontroluje se, že animace pro stání je aktivní.

☐ Ověří se, že světlo svítilny svítí.

☐ Zkontroluje se, že se postava nehýbe.

`Hráč vyskočí tlačítkem Space.`

☐ Ověří se, že světlo svítilny svítí daným směrem.

☐ Zkontroluje se, že se postava hýbe daným směrem.

`Hráč půjde doleva.`

☐ Zkontroluje se, že animace pro pohyb vlevo je aktivní.

☐ Ověří se, že světlo svítilny svítí daným směrem.

☐ Zkontroluje se, že se postava hýbe daným směrem.

`Hráč půjde doprava.`

☐ Zkontroluje se, že animace pro pohyb vpravo je aktivní.

☐ Ověří se, že světlo svítilny svítí daným směrem.

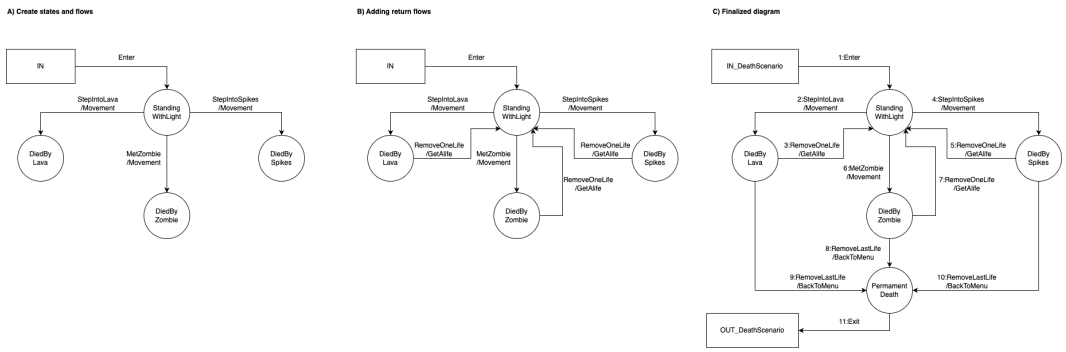
☐ Zkontroluje se, že se postava hýbe daným směrem.

`Ukončení dané úrovně smrtí, vítězstvím nebo ukončením úrovně.`

6.2.4 Diagramy pro smrt hráče a chování mince

Zbylé dva diagramy se vytvoří stejně, jako ten první. V případě diagramu pro smrt hráče se opět začne vstupem IN, který vede přes flow *Enter* do stavu *StandingWithLight*. Od něj, se rozvětní flows na tři části. První je flow *StepIntoLava* s událostí *Movement*, která vede do stavu *DiedByLava*. Druhé je flow *MetZombie* s událostí *Movement*, která vede do stavu *DiedByZombie*. Třetí je flow *StepIntoSpikes* s událostí *Movement*, která vede do stavu *DiedBySpikes*. Tento progres reprezentuje Obrázek 6.6 část A.

Obrázek 6.6 část B posouvá diagram tím, že přidává zpáteční flows. Pro stav *DiedByLava* tedy přidá flow *RemoveOneLife* s událostí *GetALife*. Další stav *DiedByZombie* bude mít to samé flow se stejnou událostí a poslední stav *DiedBySpikes* též tak. Všechny tři se vrací do stavu *StandingWithLight*.



Obrázek 6.6 Vývojový diagramu pro smrt hráče

V poslední části Obrázku 6.6 je již finální podoba diagramu. Byl přidán poslední stav *PermamentDeath* do kterého směřují všechny předešlé stavy kromě *StandingWithLight*. Pro stav *DiedByLava* je tedy přidáno flow *RemoveLastLife* s událostí *BackToMenu*. Pro stav *DiedBySpikes* a *DiedByZombie* je přidáno stejné flow se stejnou událostí. ze stavu

PermamentDeath směřuje flow *Exit* do pole OUT. Polička IN a OUT byly přejmenovány a ke každému flow bylo přidáno ID.

Vytvoří se datový slovník:

`StandingWithLight`

- ☐ Zkontroluje se, že animace pro stání je aktivní.
- ☐ Ověří se, že světlo svítilny svítí.
- ☐ Zkontroluje se, že se postava nehýbe.

`DiedByLava`

- ☐ Ověří se, že láva svítí a nehýbe se.
- ☐ Jakmile se hráč dotkne lávy, tak zemře.
- ☐ Zkontroluje se, že hráč po úmrtí je zpět na startovní pozici (pokud to nebyl poslední život).
- ☐ Zkontroluje se, že hráč po úmrtí zmizí z místa úmrtí.
- ☐ Zkontroluje se, že animace pro stání je aktivní.
- ☐ Ověří se, že světlo svítilny svítí.
- ☐ Zkontroluje se, že se postava nehýbe.

`DiedByZombie`

- ☐ Ověří se, že zombie se hýbe ve směru, v jakém má.
- ☐ Ověří se, že animace pro zombie je aktivní.
- ☐ Jakmile se hráč dotkne zombie, tak zemře.
- ☐ Zkontroluje se, že hráč po úmrtí je zpět na startovní pozici (pokud to nebyl poslední život).
- ☐ Zkontroluje se, že hráč po úmrtí zmizí z místa úmrtí.
- ☐ Zkontroluje se, že animace pro stání je aktivní.
- ☐ Ověří se, že světlo svítilny svítí.
- ☐ Zkontroluje se, že se postava nehýbe.

`DiedBySpikes`

☐ Ověří se, že hroty nic nedělají a nijak nesvítí.

☐ Jakmile se hráč dotkne hrotů, tak zemře.

☐ Zkontroluje se, že hráč po úmrtí je zpět na startovní pozici (pokud to nebyl poslední život).

☐ Zkontroluje se, že hráč po úmrtí zmizí z místa úmrtí.

☐ Zkontroluje se, že animace pro stání je aktivní.

☐ Ověří se, že světlo svítilny svítí.

☐ Zkontroluje se, že se postava nehýbe.

`IN_DeathScenario`

☐ Spuštění dané úrovně a zpřístupnění pohybu hráči.

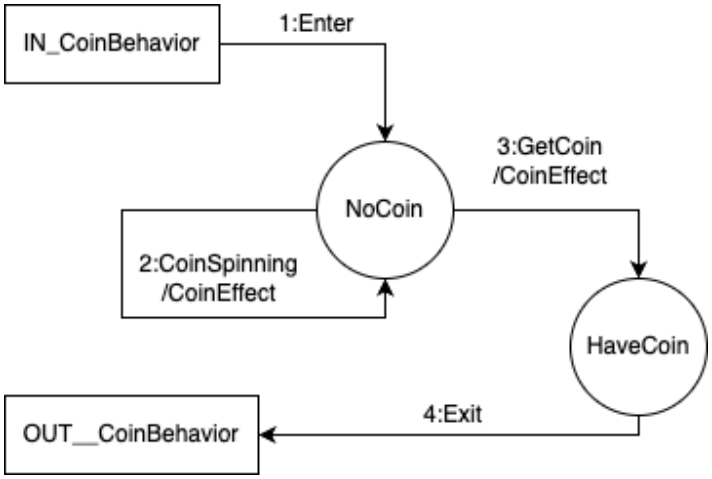
`OUT_DeathScenario`

☐ Ukončení dané úrovně smrtí, vítězstvím nebo ukončením úrovně.

Vytvoření cest:

Začne se flow 1 a přejde se do *StandingWithLight*. Následuje flow 2 do stavu *DiedByLava* a s vrácením přes flow 3 do *StandingWithLight*. Pokračuje se flow 4 do stavu *DiedBySpikes* se zpátečním flow 5 do *StandingWithLight*. Zbývá ještě flow 6 do stavu *DiedByZombie*, které se vrací flow 7 do *StandingWithLight*. Aktuálně je cesta **1,2,3,4,5,6,7**. Jako poslední je na řadě pokrýt situace s permanentní smrtí, která vrátí hráče zpět do menu. Z flow 2 přes stav *DiedByLava* a flow 9 lze dojít do stavu *PermamentDeath*. To samé platí v případě flow 6 a napojení na flow 8 anebo flow 4 a napojení na flow 10. Ze stavu *PermamentDeath* se jen napojí poslední flow do *OUT_DeathScenatio*. Konečná minimální cesta je tedy **1,2,3,4,5,6,7,2,9,11**, **1,2,3,4,5,6,7,6,8,11** nebo **1,2,3,4,5,6,7,4,10,11**. Pro případ základní cesty jsou cesty **1,2,9,11**, **1,6,8,11** nebo **1,4,10,11**.

Nakonec je potřeba vytvořit diagram pro chování mince. Z IN_CoinBehavior vychází flow s ID 1 a názvem *Enter* do stavu *NoCoin*. Stav má jednoduchou smyčku pro otáčení mince, kdy s ID 2, názvem flow *CoinSpinning* a událostí *CoinEffect* se provolává sama. Dalším flow je s ID 3, názvem *GetCoin* a událostí *CoinEffect*, která vede do stavu *HaveCoin*. V tomto případě se mince odebere z mapy, a přičte se hráči nahoře na obrazovce. Ze stavu *HaveCoin* je poslední flow *Exit* s ID 4 do OUT_CoinBehavior. Výsledný diagram je reprezentován Obrázkem 6.7.



Obrázek 6.7 Vývojový diagramu pro mince

Opět se vytvoří datový slovník:

`NoCoin`

☐ Mince se zobrazí na mapě.

`HaveCoin`

☐ Mince zmizí z obrazovky.

☐ Hráč dostane do počítadla +1 minci.

`IN_CoinBehavior`

☐ Spuštění dané úrovně a zpřístupnění pohybu hráči.

`OUT_CoinBehavior`

☐ Ukončení dané úrovně smrtí, vítězstvím nebo ukončením úrovně.

`CoinEffect`

☐ Mince má animaci otáčení.

Vytvořená cesta je ve tvaru **1,2,3,4**, kdy se jedná o minimální i základní cestu zároveň.

6.3 Cleanroom testování

V rámci vytvoření Cleanroom kombinatorické tabulky se první musely vytvořit hodnoty pro všechny atributy. Ty by se v ideálním případě měly získat studií nebo statistikou herního studia. V tomto případě byly ale napsány náhodně pro znázornění. Pro příklad se tabulka vytvoří pro typ hráče **Achiever** (hráč zaměřený na úspěchy). Tabulka bude kvůli velikosti rozdělena do více částí. To samé platí pro tabulku s procentuálními hodnotami.

Tabulka 6.5 znázorňuje první čtyři parametry s hodnotami, které charakterizují procentuální četnost využití daného nastavení.

Tabulka 6.5 Tabulka s hodnotami pro hráče zaměřeného na úspěchy (Achievera)

Map Drag Speed	Hodnota
0	10
0,9	85
1	5
Map Zoom Speed	Hodnota
0	30
1,1	60
1	10
Turns Between Autosave	Hodnota
0	33
10	30
999	37
Max Autosaves Kept	Hodnota
0	76
10	14
999	10

Pro první sloupec *Map Drag Speed* byly přes Excel a funkci *RANDBETWEEN* vygenerovány hodnoty **87, 74, 59, 99, 69, 76, 60, 55, 53**. Jelikož všechny hodnoty z Tabulky 6.5 jsou v rozmezí 11 až 85, tak všechny řádky bubou obsahovat hodnotu *0,9*. Vygenerované hodnoty druhého sloupce jsou **69, 37, 60, 14, 67, 40, 65, 78, 57**. Hodnota u čtvrtého řádku je v rozmezí 0 až 30, takže zde bude hodnota *0*. U ostatních řádků je hodnota vyšší než 30 a nižší než 61, takže se zapsalo *1,1*. Sloupec *Turns Between Autosave* dostal náhodné hodnoty **13, 76, 76, 0, 58, 23, 63, 43, 31**. V tomto případě se pro první, třetí a šestý řádek shodují hodnoty s rozmezím 0 až 33. Bude tedy zapsána hodnota *0*. Hodnota *10* je zase zapsána pro pátý, sedmý, osmý a devátý řádek, kdy náhodné hodnoty jsou v rozmezí 34 až 54. Zbylé řádky mají hodnotu *999*. Poslední řádek Tabulky 6.6 dostal náhodné hodnoty **34, 56, 38, 20, 39, 68, 6, 95, 70**. Veškeré řádky budou mít hodnotu *0* (kromě řádku osm, který je v rozmezí 91

až 100 a tudíž jeho hodnota je *999*). Kompletní tabulka pro první čtyři hodnoty je v Tabulce 6.6.

Tabulka 6.6 Cleanroom kombinatorická tabulka pro Interface settings v Civilization V (část I.)

Map Drag Speed	Map Zoom Speed	Turns Between Autosave	Max Autosaves Kept
0.9	1.1	0	0
0.9	1.1	999	0
0.9	1.1	999	0
0.9	0	0	0
0.9	1.1	10	0
0.9	1.1	0	0
0.9	1.1	10	0
0.9	1.1	10	999
0.9	1.1	10	0

Tabulka 6.7 obsahuje informace o sloupcích *Alternate Cursor Zoom Mode*, *Show All Policy Information* a *Auto Unit Cycle*. Hodnoty opět charakterizují četnost výskytu pro hráče Achievera.

U sloupce *Alternate Cursor Zoom Mode* se vygenerovaly náhodné hodnoty **38, 34, 100, 60, 65, 19, 85, 32 ,22**. Hodnota *TRUE* je nastavena u prvního, druhého, šestého, osmého a devátého řádku. Hodnota *FALSE* je nastavena u zbývajících řádků, protože náhodné hodnoty řádků byly v rozmezí 51 až 100. Druhý sloupec *Show All Policy Information* má náhodné hodnoty **74, 46, 12, 73, 70, 28, 34, 39, 14**. Zde byla pro hodnotu *TRUE* rozpětí pouze 0 až 9. Jelikož všechny hodnoty jsou větší než 9, tak všechny řádky nesou hodnotu *FALSE*. A pro poslední sloupec *Auto Unit Cycle* jsou hodnoty **30, 60, 63, 20, 29, 6 ,100, 76, 79**. Poslední tři řádky nesou hodnotu *FALSE* protože jsou v rozmezí 68 až 100. U zbylých řádků je vloženo *TRUE*. Hotová Tabulka je 6.8.

Tabulka 6.7 Tabulka s hodnotami pro hráče zaměřeného na úspěchy (Achievera)

Alternate Cursor Zoom Mode	Hodnota
TRUE	50
FALSE	50
Show All Policy Information	Hodnota
TRUE	9
FALSE	91
Auto Unit Cycle	Hodnota
TRUE	67
FALSE	33

Tabulka 6.8 Cleanroom kombinatorická tabulka pro Interface settings v Civilization V (část II.)

Alternate Cursor Zoom Mode	Show All Policy Information	Auto Unit Cycle
TRUE	FALSE	TRUE
TRUE	FALSE	TRUE
FALSE	FALSE	TRUE
FALSE	FALSE	TRUE
FALSE	FALSE	TRUE
TRUE	FALSE	TRUE
FALSE	FALSE	FALSE
TRUE	FALSE	FALSE
TRUE	FALSE	FALSE

Tabulka 6.9 obsahuje informace o sloupcích *Single Player Score List*, *Multiplayer Score List* a *Enable Map Inertia*. Hodnoty opět charakterizují četnost výskytu pro hráče Achievera.

Tabulka 6.9 Tabulka s hodnotami pro hráče zaměřeného na úspěchy (Achievera)

Single Player Score List	Hodnota
TRUE	20
FALSE	80
Multiplayer Score List	Hodnota
TRUE	24
FALSE	76
Enable Map Inertia	Hodnota
TRUE	95
FALSE	5

Sloupec *Single Player Score List* obsahuje vygenerované hodnoty **34, 93, 64, 97, 96, 64, 71, 83, 72**. Jelikož všechny hodnoty jsou v vyšší než 20, tak všechny řádky obsahují *FALSE*. *Multiplayer Score List* měl hodnoty **24, 5, 0, 33, 97, 91, 21, 85, 54**. Pro první, druhý, třetí a sedmý řádek je hodnota *TRUE* (Vygenerované čísla byly v rozmezí 0 až 24). Zbylé řádky vlastní *FALSE*. Pro poslední sloupec *Enable Map Inertia* Tabulky 6.10 jsou náhodné hodnoty **55, 63, 99, 18, 36, 41, 73, 26, 75**. Zde pouze třetí řádek splňuje rozmezí 95 až 100, které platí pro hodnotu *FALSE*. Všech osm zbývajících řádků má hodnotu *TRUE*.

Tabulka 6.10 Cleanroom kombinatorická tabulka pro Interface settings v Civilization V (část III.)

Single Player Score List	Multiplayer Score List	Enable Map Inertia
FALSE	TRUE	TRUE
FALSE	TRUE	TRUE
FALSE	TRUE	FALSE
FALSE	FALSE	TRUE
FALSE	FALSE	TRUE
FALSE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	TRUE
FALSE	FALSE	TRUE

V Tabulce 6.11 jsou popsány poslední tři sloupce a jejich nastavení pro náhodné hodnoty.

Tabulka 6.11 Tabulka s hodnotami pro hráče zaměřeného na úspěchy (Achievera)

Skip Intro Video	Hodnota
TRUE	87
FALSE	13
Automatically Size Interface	Hodnota
TRUE	56
FALSE	44
Use Small Scale Interface	Hodnota
TRUE	55
FALSE	45

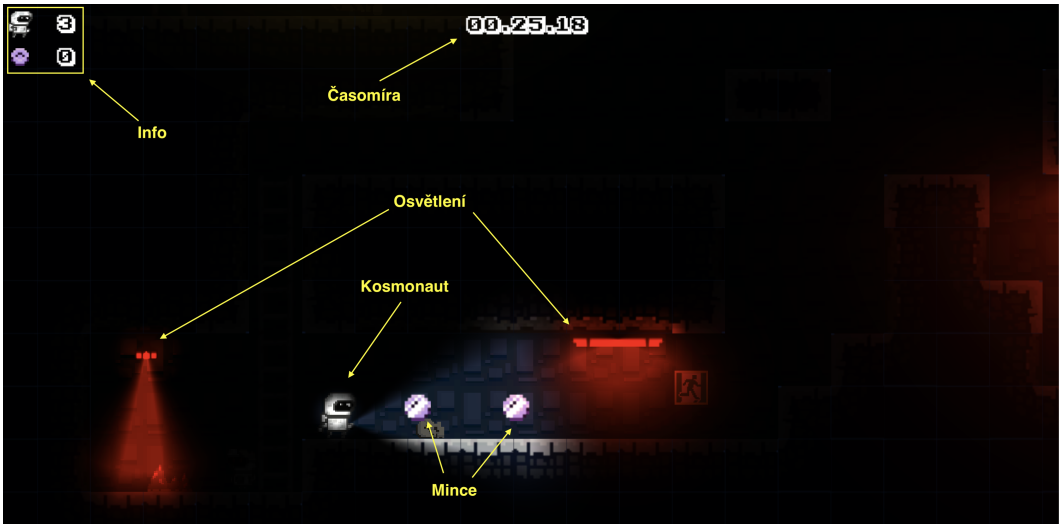
U sloupce *Skip Intro Video* jsou vygenerovány tyto náhodné hodnoty **62, 49, 31, 42, 25, 75, 62, 99, 32**. Pro zapsání hodnoty *TRUE* je potřeba splnit rozmezí 0 až 87. Toto splnil první, třetí, pátý, šestý, sedmý, osmý a devátý řádek. Pro zbývající řádky byla zapsána hodnota *FALSE*. Vygenerované hodnoty **43, 91, 62, 54, 59, 61, 30, 72, 59** patří druhému sloupci *Automatically Size Interface*. V případě rozmezí 0 až 56 byly řádky dva, tři a osm. Těm tedy patří hodnota *TRUE*. Pro zbylé je hodnota *FALSE*. Pro poslední sloupec *Use Small Scale Interface* jsou hodnoty **2, 65, 61, 53, 89, 13, 30, 22, 76**. Hodnot *FALSE* je celkem 4 a patří na druhý, třetí, pátý a devátý řádek. Pouze u nich se splnila podmínka, že hodnoty jsou v rozmezí 56 až 100. Zbylé hodnoty jsou rovny *TRUE*. Kompletní Tabulka je 6.12.

Tabulka 6.12 Cleanroom kombinatorická tabulka pro Interface settings v Civilization V (část IV.)

Skip Intro Video	Automatically Size Interface	Use Small Scale Interface
TRUE	FALSE	TRUE
FALSE	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	TRUE
TRUE	FALSE	FALSE
TRUE	FALSE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE

7 FUNKCIONÁLNÍ TESTOVÁNÍ

Pro funkcionální testy se využije hra na Obrázku 7.1. Na screenshotu jsou části, kterých se testy týkají: infopanel, časomíra, kosmonaut, osvětlení a mince.



Obrázek 7.1 Ukázka vyvíjené hry

Scénář: Testování pohybu kosmonauta

1. Kosmonaut se pohybuje doprava:
   (a) Hráč stiskne odpovídající klávesu nebo ovládací prvek (např. klávesa D nebo šipka doprava).
   (b) Ověří se, že kosmonaut plynule pohne doprava po stisknutí klávesy.
   (c) Zkontroluje se, zda je pohyb doprava bezproblémový a bez zpoždění.
2. Kosmonaut se pohybuje nahoru:
   (a) Hráč stiskne odpovídající klávesu nebo ovládací prvek (klávesa mezerník).
   (b) Ověří se, že kosmonaut plynule pohne nahoru po stisknutí klávesy.
   (c) Zkontroluje se, zda je pohyb nahoru bezproblémový a bez zpoždění.
3. Kosmonaut se pohybuje doleva:
   (a) Hráč stiskne odpovídající klávesu nebo ovládací prvek (např. klávesa A nebo šipka doleva).
   (b) Ověří se, že kosmonaut plynule pohne doleva po stisknutí klávesy.
   (c) Zkontroluje se, zda je pohyb doleva bezproblémový a bez zpoždění.
4. Reakce na vstup hráče:

(a) Hráč interaguje s klávesnicí nebo ovladačem.

(b) Ověří se, že kosmonaut reaguje okamžitě na vstup hráče a pohybuje se podle zadání.

(c) Zkontroluje se, zda je reakce na vstup hráče plynulá a bezproblémová.

(d) Svítilna je namířena směrem, kterým je namířen kosmonaut.

Scénář: Testování sbírání mincí

1. Rozmístění mincí:

   (a) Hra je spuštěna a mapa se načte.

   (b) Ověří se, že mince jsou náhodně rozmístěny po mapě.

   (c) Zkontroluje se, zda mince nejsou umístěny na nedostupných místech nebo v pastech.

2. Zobrazení aktuálního počtu mincí:

   (a) Hráč se dotkne mince.

   (b) Ověří se, že se po sebrání mince aktualizuje aktuální počet mincí na obrazovce.

   (c) Zkontroluje se, zda je aktuální počet mincí zobrazován správně.

3. Reakce na sebrání všech mincí:

   (a) Hráč sebere všechny mince na mapě.

   (b) Ověří se, že hra reaguje na sebrání všech mincí.

   (c) Zkontroluje se, zda hra ukončí úroveň nebo zobrazí upozornění, že všechny mince byly nasbírány.

Testování osvětlení:

1. Viditelnost ve tmě:

   (a) Hra je spuštěna a mapa se načte.

   (b) Ověří se, zda je ve hře patrná tma.

   (c) Zkontroluje se, zda světlo od svítilny funguje správně a osvětluje okolí hráče.

2. Problikávání světel:

   (a) Hra je spuštěna a mapa se načte.

(b) Ověří se, zda některá světla na mapě problikávají.

(c) Zkontroluje se, jak problikávání světel ovlivňuje viditelnost hráče a jeho schopnost navigace.

Testování pastí:

1. Následky pádu do pasti:

   (a) Hráč spadne do pasti, například na ostny, do lávy nebo narazí na zombie kosmonauta.

   (b) Ověří se, že hráč umírá po kontaktu s pastí.

   (c) Zkontroluje se, zda hra reaguje na smrt hráče a zda ukončí úroveň nebo vrátí hráče na začátek.

Testování času:

1. Sledování času:

   (a) Hra je spuštěna a úroveň se načte.

   (b) Ověří se, zda hra sleduje čas a zda má určený časový limit pro dokončení úrovně.

   (c) Zkontroluje se, zda je čas zobrazen hráči a aktualizuje se v průběhu hry.

   (d) Jakmile zbývá méně než 10 sekund, časomíra zčervená.

2. Reakce na nedokončení úrovně v časovém limitu:

   (a) Hráč nedokončí úroveň v časovém limitu.

   (b) Ověří se, jak hra reaguje na nedokončení úrovně v daném časovém limitu.

   (c) Zkontroluje se, zda hra ukončí úroveň a zobrazí upozornění o neúspěchu.

Testování obecné hratelnosti:

1. Možnost začít novou hru:

   (a) Hráč dokončí úroveň nebo umře.

   (b) Ověří se, zda hráč může začít novou hru po dokončení nebo smrti.

   (c) Zkontroluje se, zda hra přepne hráče zpět na začátek bez problémů.

2. Přepínání mezi úrovněmi:

(a) Hráč dokončí úroveň a postoupí na další.

(b) Ověří se, zda hráč může přepnout mezi úrovněmi, pokud jsou k dispozici.

(c) Zkontroluje se, zda je přepínání mezi úrovněmi plynulé a bezproblémové.

3. Správné fungování menu hry:

(a) Hráč se pokusí přistoupit k menu hry.

(b) Ověří se, zda je menu hry přístupné a zda všechny prvky fungují správně.

(c) Zkontroluje se, zda hráč může spustit novou hru, načíst uloženou hru nebo odejít z hry.

7.1 Unit testování

Testování je rozděleno na PlayMode a EditMode. Rozdíl je, že PlayMode testy se spouští se samotnou hrou (imitují hraní), zatímco EditMode testuje mimo samotné hraní. Každý set testů je rozdělen do souborů, které jsou pojmenovány podle script souborů, které testují.

7.1.1 TimerTests testy

Testy sloužící ke třídě Timer, která implementuje časovač s náhodně generovanou dobou, se kterou se počítá od maximálního do minimálního času. Po startu časovač běží a zobrazuje zbylý čas v textovém poli. Pokud časovač dosáhne nuly, zobrazí se **00.00.00** a časovač se zastaví.

```
1     private GameObject timerGameObject;
2     private Timer timer;
3
4     [SetUp]
5     public void Setup()
6     {
7         // Create a new game object and add the Timer component to it
8         timerGameObject = new GameObject();
9         timer = timerGameObject.AddComponent<Timer>();
10    }
11
12
13    [TearDown]
14    public void Teardown()
15    {
16        // Destroy the game object after each test
17        GameObject.DestroyImmediate(timerGameObject);
18    }
19
20    [UnityTest]
21    public IEnumerator
      TimerChangesTextColorWhenTimerIsBelowOrEqualTo10()
22    {
23        // Arrange
24        float timerBelow10 = 9f;
25        TextMeshProUGUI timerText = timerGameObject.AddComponent<
      TextMeshProUGUI>();
26        timer.timerText = timerText;
27
28        // Act
29        timer.DisplayTime(timerBelow10);
30
31        // Wait for one frame to ensure UI updates
32        yield return null;
33
34        // Assert
```

```
35          Assert.AreEqual(Color.red, timerText.color); // Text color
    should be red when timer is below or equal to 10
36      }
```

Tento kód obsahuje třídu, která obsahuje sadu unit testů pro třídu *Timer*. Nejprve jsou inicializovány proměnné pro herní objekt a instanci třídy *Timer*. Metoda *Setup* vytváří nový herní objekt a připojuje k němu komponentu *Timer*. Metoda se spouští vždy před každým testem. Metoda *Teardown* je použita k odstranění herního objektu po dokončení každého testu. Metoda se spouští vždy po každém testu. Jednotlivé testy jsou implementovány jako metody označené atributem *UnityTest*. Každý test provádí určitou akci a následně ověřuje očekávané chování.

Test *TimerChangesTextColorWhenTimerIsBelowOrEqualTo10* ověřuje, zda se barva textu změní na červenou, když hodnota časovače je nižší nebo rovna 10. Nejprve v přípravném kroku (Arrange) je nastavena hodnota *timerBelow10* na 9 sekund a k hernímu objektu *timerGameObject* je přidána komponenta *TextMeshProUGUI*, která slouží jako zobrazení časovače. Dále je nastavena vlastnost *timerText* instance třídy *Timer* na právě přidanou komponentu *TextMeshProUGUI*, abychom mohli později ověřit barvu textu. V akčním kroku (Act) je zavolána metoda *DisplayTime* instance třídy *Timer* s parametrem *timerBelow10*, což simuluje zobrazení času 9 sekund. Počkáme na jedno snímání obrazovky (Wait), což zajišťuje aktualizaci uživatelského rozhraní. V posledním kroku (Assert) ověřujeme, zda se barva textu změnila na červenou pomocí metody *Assert.AreEqual*. Pokud je hodnota časovače nižší nebo rovna 10, měla by být barva textu nastavena na červenou.

```
1       [UnityTest]
2       public IEnumerator TimerDecreaseTimerAndUpdatesText()
3       {
4           // Arrange
5           TextMeshProUGUI timerText = timerGameObject.AddComponent<
    TextMeshProUGUI>();
6           timer.timerText = timerText;
7           timer.Start();
8
9           // Act
10          yield return new WaitForSeconds(1);
11
12          // Assert
13          Assert.IsTrue(timer.timer < timer.randomNum - 1);
14          Assert.AreNotEqual("00.00.00", timerText.text);
15      }
```

Test *TimerDecreaseTimerAndUpdatesText* ověřuje, zda se časovač snižuje a zda se aktualizuje textové zobrazení časovače. V přípravném kroku (Arrange) se nejprve přidá komponenta *TextMeshProUGUI* k hernímu objektu *timerGameObject*, který slouží jako zobrazení časovače. Následně nastavíme vlastnost *timerText* instance třídy *Timer* na právě přidanou komponentu *TextMeshProUGUI*, abychom mohli později ověřit aktualizaci textového zobrazení. Dále spustíme časovač voláním metody *Start* instance třídy *Timer*, což simuluje jeho běh. V akčním kroku (Act) se počká jednu sekundu pomocí *yield return new WaitForSeconds(1);*. Tím se simuluje uplynutí jedné sekundy, což je typický časový úsek mezi aktualizacemi časovače. V posledním kroku (Assert) se pak

ověří, zda se hodnota časovače skutečně snížila a zda se změnilo i textové zobrazení časovače. To provádíme pomocí metody *Assert.IsTrue*, která porovnává hodnotu *timer.timer* s hodnotou *timer.randomNum - 1* a zajišťuje, že časovač klesl o více než jednu sekundu. Dále ověřujeme, že textové zobrazení časovače se změnilo z výchozí hodnoty "00.00.00", což by mohlo indikovat aktualizaci textu.

```
1     [UnityTest]
2     public IEnumerator TimerCheckIfTimerTextIsNotNull()
3     {
4         // Arrange
5         TextMeshProUGUI timerText = timerGameObject.AddComponent<
    TextMeshProUGUI>();
6         timer.timerText = timerText;
7         timer.Start();
8
9         // Act
10        yield return new WaitForSeconds(1);
11
12        // Assert
13        Assert.IsNotNull(timerText);
14    }
```

Tento test ověřuje, zda není textové zobrazení časovače *null*, tj. zda je správně inicializováno a přiřazeno třídě *Timer*. V přípravném kroku (Arrange) se nejprve přidá komponenta *TextMeshProUGUI* k hernímu objektu *timerGameObject*, který slouží jako zobrazení časovače. Následně se nastaví vlastnost *timerText* instance třídy *Timer* na právě přidanou komponentu *TextMeshProUGUI*, abychom mohli ověřit její existenci. Dále se spustí časovač voláním metody *Start* instance třídy *Timer*, což simuluje jeho běh. V akčním kroku (Act) se počká jednu sekundu pomocí *yield return new WaitForSeconds(1);*. Tím se simuluje uplynutí jedné sekundy, což je typický časový úsek mezi aktualizacemi časovače. V posledním kroku (Assert) se ověří, zda textové zobrazení časovače není *null*. To se provede pomocí metody *Assert.IsNotNull*. Pokud by textové zobrazení bylo null, test by selhal.

```
1     [UnityTest]
2     public IEnumerator TimerDoesNotGoNegative()
3     {
4         // Arrange
5         TextMeshProUGUI timerText = timerGameObject.AddComponent<
    TextMeshProUGUI>();
6         timer.timerText = timerText;
7         timer.Start();
8         timer.randomNum = 2f;
9
10        // Act
11        yield return new WaitForSeconds(timer.randomNum + 2);
12
13        // Assert
14        Assert.GreaterOrEqual(timer.timer, 0f);
15    }
```

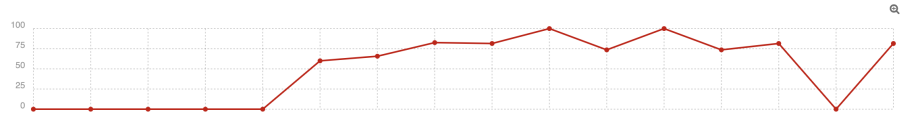
Test *TimerDoesNotGoNegative* ověří, že časovač se nikdy nezobrazí s negativní hodnotou. V přípravném kroku (Arrange) je nejprve přidána komponenta *TextMeshProUGUI* k hernímu objektu *timerGameObject*, který slouží jako zobrazení časovače. Následně je nastavena vlastnost *timerText* instance třídy *Timer* na právě přidanou komponentu *TextMeshProUGUI*, abychom mohli ověřit, že časovač nezobrazuje negativní hodnoty. Dále je spuštěn časovač voláním metody *Start* instance třídy *Timer*, což simuluje jeho běh, a je nastavena hodnota *randomNum* na 2 sekundy. V akčním kroku

(Act) se čeká na časový úsek delší než *randomNum* plus 2 sekundy. Tím se zajistí, že časovač již skončil a měl by ukázat nulu. V posledním kroku (Assert) se následně ověří, že hodnota časovače je větší nebo rovna nule pomocí metody *Assert.GreaterOrEqual*. Pokud by časovač měl negativní hodnotu, test by selhal.

Na konci se vytvoří Code Coverage analýza, která ukáže pokrytí kódu, který je otestován. Na Obrázku 7.2 je výňatek, který popisuje hlavní aspekty jako Třídu, umístění, počet řádků, počet pokrytých řádků, počet nepokrytých řádků a procentuální zastoupení pokrytí testy. Je k dispozici i diagram s historií pokrytí a seznam metod, které jsou a které nejsou pokryty.

Class:	Timer
Assembly:	Scripts
File(s):	/Users/zdenek/Unity/Aurora 2/Assets/Scripts/Timer.cs
Covered lines:	31
Uncovered lines:	7
Coverable lines:	38
Total lines:	63
Line coverage:	81.5% (31 of 38)
Covered branches:	0
Total branches:	0

Coverage History



Metrics

Method	Branch coverage	Crap Score	Cyclomatic complexity	NPath complexity	Sequence coverage
Timer()	0%	0	0	0	100%
Start()	0%	0	0	0	100%
Update()	0%	0	0	0	58.82%
DisplayTime(...)	0%	0	0	0	100%

Obrázek 7.2 Code coverage pro třídu Timer

Obrázek 7.3 vyobrazuje řádky kódu, které nejsou pokryty. Jedná se o podmínku s proměnnou *IsItCounting* a její částí, kdy podmínka skončí v else a *IsItCounting* = *false*. Tento případ se již komplikovaněji testuje, protože proměnná je private ve třídě *Timer* a k ní nemá unit testování přístup. Na druhou stranu se toto chování, kdy timer nemůže jít do záporu testuje v *TimerDoesNotGoNegative*.

```
        25    // Update is called once per frame
        26    void Update()
      4372    27    {
      4372    28        if(isItCounting)
      4372    29        {
      4372    30            if(timer >= 0)
      4372    31            {
      4372    32                timer -= Time.deltaTime;
      4372    33                DisplayTime(timer);
      4372    34            } else
         0    35            {
         0    36                timerText.text = "00.00.00";
         0    37                isItCounting = false;
         0    38            }
      4372    39        } else
         0    40        {
         0    41            isItCounting = false;
         0    42        }
      4372    43    }
```

Obrázek 7.3 Code coverage pro třídu Timer

7.1.2 EnemyMovementTests testy

Třída *Enemy_movement* řídí pohyb nepřítele vpřed nebo vzad podle směru, kterým je otočen. Otočení se provádí po opuštění kolizního objektu. Soubor řeší testy přímo pro tuto třídu a její metody.

```
    [UnityTest]
    public IEnumerator EnemyMovesLeftWhenFacingLeft()
    {
        // Arrange
        GameObject enemyObject = new GameObject();
        enemyObject.AddComponent<Rigidbody2D>();
        Enemy_movement enemy = enemyObject.AddComponent<Enemy_movement
>();
        enemy.moveSpeed = 1f;
        enemyObject.transform.localScale = new Vector2(1, 1);

        // Act
        yield return null;

        // Assert
        Assert.AreEqual(1f, enemy.GetComponent<Rigidbody2D>().velocity
.x);
    }
```

Test *EnemyMovesLeftWhenFacingLeft* testuje správnou funkci pohybu nepřítele doleva v situaci, kdy je nepřítel otočen doleva. V přípravném kroku (Arrange) se nejprve vytvoří nový herní objekt reprezentující nepřítele. K tomuto objektu se přidá komponentu *Rigidbody2D* pro simulaci fyzikálního chování. Dále se k tomuto objektu přidá komponenta *Enemy_movement*, která je testována. Nastaví se pohybová rychlost nepřítele na hodnotu *1f* a nepřítel se otočí doleva nastavením jeho měřítka na hodnotu *(1, 1)*. V kroku akce (Act) se počká na uplynutí jednoho snímku obrazovky pomocí *yield return null;*. Tento krok simuluje časový úsek, během kterého může být aplikován pohyb. V posledním kroku (Assert) se pak ověří, zda má nepřítel nastavenou rychlost pohybu na hodnotu *1f* ve směru doleva. To je provedeno porovnáním hodnoty *velocity.x* komponenty *Rigidbody2D* nepřítele s očekávanou hodnotou *1f* pomocí metody *Assert.AreEqual*.

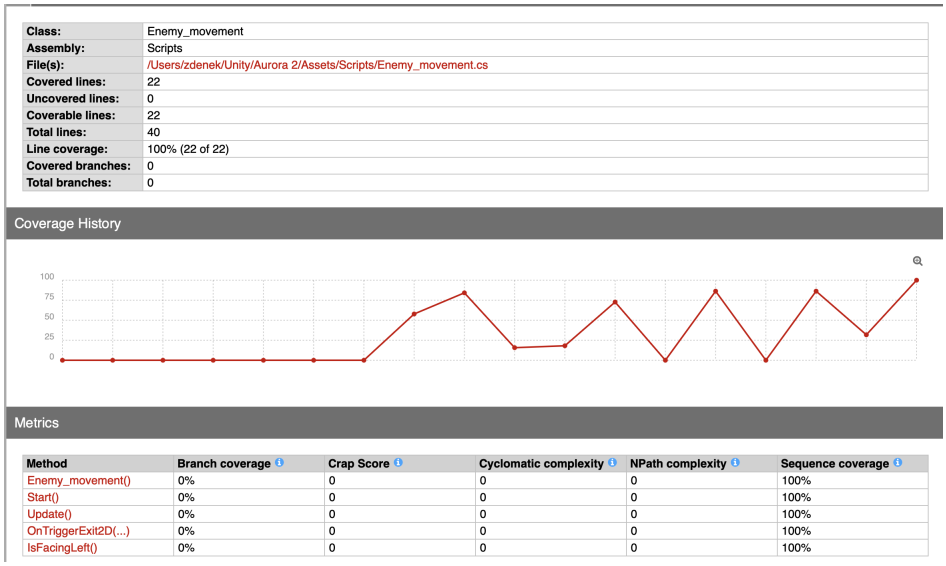
```
    [UnityTest]
    public IEnumerator EnemyMovesRightWhenFacingRight()
    {
        // Arrange
        GameObject enemyObject = new GameObject();
        enemyObject.AddComponent<Rigidbody2D>();
        Enemy_movement enemy = enemyObject.AddComponent<Enemy_movement
>();
        enemy.moveSpeed = 1f;
        enemyObject.transform.localScale = new Vector2(-1, 1);

        // Act
        yield return null;

        // Assert
        Assert.AreEqual(-1f, enemy.GetComponent<Rigidbody2D>().
velocity.x);
    }
```

Obdobný test jako *EnemyMovesLeftWhenFacingLeft* s tím rozdílem, že se kontroluje otočení nepřítele směrem doprava. Pohybová rychlost se nastaví na *-1f* a poté se kontroluje, jestli má nepřítel tuto hodnotu nastavenou v komponentě *Rigidbody2D* ve *ve-*

locity.x.

```
    [UnityTest]
    public IEnumerator EnemyChangesDirectionWhenExitingTrigger()
    {
        // Arrange
        GameObject enemyObject = new GameObject();
        Rigidbody2D rb = enemyObject.AddComponent<Rigidbody2D>();
        Enemy_movement enemy = enemyObject.AddComponent<Enemy_movement
>();
        enemy.moveSpeed = 1f;
        enemyObject.transform.localScale = new Vector2(1, 1); //
Initially facing left

        enemy.Start();

        // Act
        enemy.OnTriggerExit2D(null);

        // Assert
        Assert.AreEqual(1f, Mathf.Sign(rb.velocity.x));
        yield return null;
    }
```

Metoda *EnemyChangesDirectionWhenExitingTrigger* testuje správné chování nepřítele při opuštění kolizního objektu. V přípravném kroku (Arrange) se nejprve vytvoří nový herní objekt reprezentující nepřítele. K tomuto objektu se přidá komponenta *Rigidbody2D* pro simulaci fyzikálního chování. Dále se k tomuto objektu přidá komponenta *Enemy_movement*, která je testována. Nastaví se pohybová rychlost nepřítele na hodnotu *1f* a nastaví se jeho orientace na počáteční hodnotu, tj. otočení doleva. Následně se volá metoda *Start()* instance třídy *Enemy_movement*, která inicializuje referenci na *Rigidbody2D*. V kroku akce (Act) se simulujeme opuštění kolizního objektu voláním metody *OnTriggerExit2D(null)*. Tím se spouští metoda, která změní směr pohybu nepřítele na opačný. V posledním kroku (Assert) se ověří, že směr pohybu nepřítele po opuštění kolizního objektu je správně nastaven na *1f*, což odpovídá pohybu doprava. Nakonec se počká na jeden snímek obrazovky pomocí *yield return null;*, aby se mohlo provést asynchronní testování.



Class:	Enemy_movement
Assembly:	Scripts
File(s):	/Users/zdenek/Unity/Aurora 2/Assets/Scripts/Enemy_movement.cs
Covered lines:	22
Uncovered lines:	0
Coverable lines:	22
Total lines:	40
Line coverage:	100% (22 of 22)
Covered branches:	0
Total branches:	0

Coverage History

Metrics

Method	Branch coverage	Crap Score	Cyclomatic complexity	NPath complexity	Sequence coverage
Enemy_movement()	0%	0	0	0	100%
Start()	0%	0	0	0	100%
Update()	0%	0	0	0	100%
OnTriggerExit2D(...)	0%	0	0	0	100%
IsFacingLeft()	0%	0	0	0	100%

Obrázek 7.4 Code coverage pro třídu Enemy_movement

Na Obrázku 7.4 je analýza pokrytí kódu testy pro třídu *Enemy_ movement*. Tentokrát je pokryto všech 22 řádků, které je možné pokrýt. Je vhodné zmínit, že pokrytí řádků nutně neznamená, že jsou pokryty všechny UC.

8 NEFUNKCIONÁLNÍ TESTOVÁNÍ

Pro Nefunkcionální testování je klíčové zlepšení použitelnosti, efektivity a udržovatelnosti. Následující kapitola se soustředí v první části na výkonnostní testování, kde se za pomoci testů bude zkoušet snímková frekvence a rychlost načítání scén. Poté ve druhé části se zaměří na testování použitelnosti, které je spíše souborem pravidel a postupů, jak by se mělo správně testovat v případě využití externích účastníků.

8.1 Výkonnostní testování

Výkonnostní testování v Unity bylo zaměřeno na zajištění optimálního výkonu a uživatelské zkušenosti na rozpracovaném projektu hry. Pro účely automatizovaných testů se často používá nástroj Test Runner. Naopak pro účely manuálního testování se využívá nástroj Simulator a Profiler (je ovšem možné je kombinovat dohromady spolu s automatizovanými testy). Zmíněné testy v kapitole se zaměřují na rychlost načítání scén a počet snímků za sekundu (FPS), což jsou klíčové faktory ovlivňující plynulost a reaktivitu hry.

```
1     private int firstSceneIndex = 0; // Scene Index Main Menu
2     private int secondSceneIndex = 1; // Scene Index of game
3     private float maxLoadingTime = 5f;
4
5     [UnityTest]
6     public IEnumerator TestSceneLoadingSpeed()
7     {
8         float firstSceneLoadingTime = 0f;
9         float secondSceneLoadingTime = 0f;
10
11        AsyncOperation firstSceneLoadOperation = SceneManager.
    LoadSceneAsync(firstSceneIndex, LoadSceneMode.Additive);
12
13        float startTime = Time.realtimeSinceStartup;
14        yield return firstSceneLoadOperation;
15        firstSceneLoadingTime = Time.realtimeSinceStartup - startTime;
16
17        Debug.Log("firstSceneLoadingTime: " + firstSceneLoadingTime);
18        Assert.LessOrEqual(firstSceneLoadingTime, maxLoadingTime, "The
     first scene took longer to load than the allowed limit.");
19
20        yield return new WaitForSeconds(0.5f);
21
22        SceneManager.UnloadSceneAsync(firstSceneIndex);
23
24        AsyncOperation secondSceneLoadOperation = SceneManager.
    LoadSceneAsync(secondSceneIndex, LoadSceneMode.Additive);
25
26        startTime = Time.realtimeSinceStartup;
27        yield return secondSceneLoadOperation;
28        secondSceneLoadingTime = Time.realtimeSinceStartup - startTime
    ;
29
30        Debug.Log("secondSceneLoadingTime: " + secondSceneLoadingTime)
    ;
31
32        Assert.LessOrEqual(secondSceneLoadingTime, maxLoadingTime, "
    The second scene took longer to load than the allowed limit.");
33
34        SceneManager.UnloadSceneAsync(secondSceneIndex);
35    }
```

Test *TestSceneLoadingSpeed* byl vytvořen k měření rychlosti načítání dvou scén v Unity a následnému ověření, zda čas načítání každé scény nepřekročil stanovený limit. Test začíná načtením scény hlavního menu pomocí funkce *LoadSceneAsync* s předáním indexu hlavní scény. Následně je uložen aktuální čas pomocí *Time.realtimeSinceStartup*.

Poté následuje čekání na dokončení načítání scény, což je realizováno pomocí příkazu *yield return firstSceneLoadOperation*. Po dokončení načítání je opětovně uložen aktuální čas a z něj je odečten čas před spuštěním načítání. Tato operace vytváří proměnnou *firstSceneLoadingTime*, která obsahuje časový údaj o tom, jak dlouho trvalo načítání scény. Další řádek testu obsahuje kontrolu pomocí *Assert.LessOrEqual*, která ověřuje, zda čas načítání první scény nepřekročil stanovený limit. Pokud testování této podmínky selže, test ukončí svůj běh a zobrazí se chybové hlášení. Poté je test přesunut do druhé části, kde probíhá obdobný proces pro načítání druhé scény, tentokrát s použitím indexu druhé scény. Po dokončení načítání se opětovně měří čas a kontroluje se, zda nepřekročil stanovený limit. Nakonec jsou obě scény odstraněny voláním *UnloadSceneAsync*. Tím se zajišťuje čistota prostředí pro následující testy a minimalizuje se potenciální vliv na výsledky.

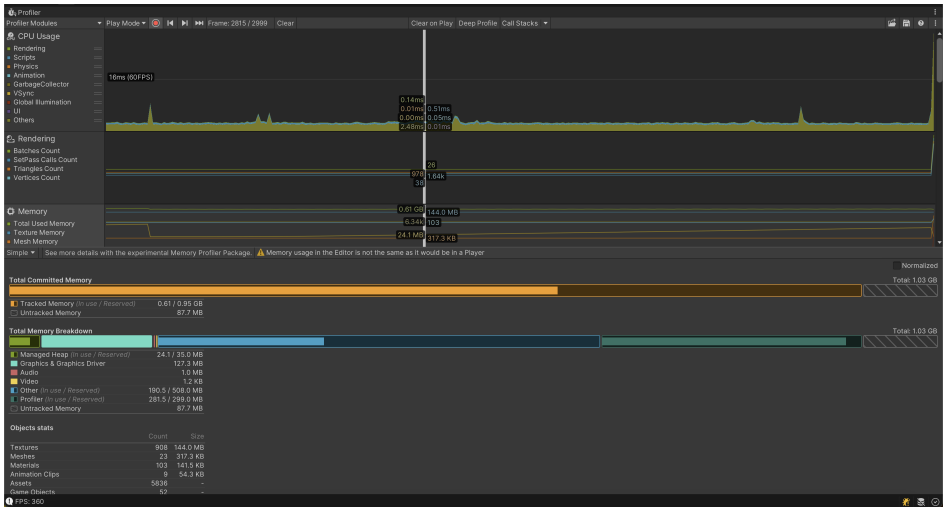
```
    private float testDuration = 10f;
    private float minFPS = 30f;
    private float fpsThreshold = 5f;

    [UnityTest]
    public IEnumerator FPSTestWithoutYield()
    {
        float elapsedTime = 0f;
        float totalFrames = 0f;

        yield return null;

        float startTime = Time.realtimeSinceStartup;

        while (elapsedTime < testDuration)
        {
            totalFrames++;

            elapsedTime = Time.realtimeSinceStartup - startTime;
        }

        float averageFPS = totalFrames / testDuration;

        Assert.GreaterOrEqual(averageFPS, minFPS, "The average FPS is lower than the minimum acceptable value.");
    }
```

Tento test byl navržen k ověření průměrného počtu snímků za sekundu (FPS) během určité doby trvání testu v Unity. Cílem je zkontrolovat, zda průměrná FPS během testu dosahuje nebo překračuje stanovenou minimální hodnotu. Test začíná inicializací proměnných *elapsedTime* a *totalFrames*, které slouží k sledování uplynulého času a celkového počtu snímků. Poté následuje *yield return null*, který zajišťuje, že test začne až v následujícím snímku hry. Po této inicializaci začne smyčka while, která běží, dokud neuplyne stanovená doba testu. V každé iteraci smyčky se zvýší počet snímků *totalFrames* a aktualizuje se uplynulý čas *elapsedTime*. Po skončení testu se vypočítá průměrná FPS jako podíl celkového počtu snímků a délky trvání testu. Tato hodnota je porovnána s minimální přijatelnou FPS pomocí *Assert.GreaterOrEqual*. Pokud průměrná FPS nedosáhne stanoveného minima, test selže a zobrazí se odpovídající chybové hlášení. Tímto způsobem testuje tento test, zda je výkon hry dostatečný vzhledem k minimálním požadavkům na FPS.

```
    private float testDuration = 10f;
    private float minFPS = 30f;
    private float fpsThreshold = 5f;
```

```
4
5       [UnityTest]
6       public IEnumerator FPSTestWithYield()
7       {
8           float elapsedTime = 0f;
9
10          yield return null;
11
12          while (elapsedTime < testDuration)
13          {
14              float currentFPS = 1f / Time.deltaTime;
15
16              Assert.GreaterOrEqual(currentFPS, fpsThreshold, "FPS is
    lower than the minimum acceptable value.");
17
18              elapsedTime += Time.deltaTime;
19
20              yield return null;
21          }
22      }
```

Alternativně lze použít i tento test. Oba testy mají své využití v závislosti na požadovaném chování a metodě měření FPS. *FPSTestWithoutYield* je vhodný pro testování průměrné FPS v průběhu delšího časového úseku, zatímco *FPSTestWithYield* umožňuje detailnější sledování FPS v reálném čase během testu.



Obrázek 8.1 Unity Profiler

Na Obrázku 8.1 je znázorněn nástroj Profiler pro Unity. Využívá se pro analýzu a živé měření výkonu hry. Lze v něm měřit CPU a GPU usage, zatížení paměti a vykreslování, zatížení při spuštění hudby nebo videa atd. Na snímku lze vidět zatížení CPU, které zobrazovalo nad 60 FPS (snímků za sekundu) a v detailu je zobrazeno využití paměti.

8.2 Testování použitelnosti

Testování použitelnosti umožňuje získat cennou zpětnou vazbu od skutečných uživatelů a optimalizovat uživatelský zážitek. Tento proces se skládá ze čtyř hlavních podkapitol, které zahrnují přípravu a stanovení cílů, výběr vhodných hráčů, průběh testování a analýzu a zpracování výsledků (viz teoretická část). Každá z těchto fází hraje důležitou

roli při identifikaci nedostatků a možností zlepšení. Tato kapitola je více teoretická, ale dostatečně aplikovatelná.

8.2.1 Příprava a stanovení cílů

Je dobré si založit seznam cílů, které je potřeba zajistit v rámci testování. V případě přípravy pro otestování rozpracované hry by se mohlo jednat o tyto cíle:

1. *Zhodnotit přívětivost a intuitivnost hlavního menu:* Zjistit, zda je hlavní menu snadno navigovatelné a zda hráči snadno najdou potřebné funkce, jako jsou možnosti spuštění hry, přístup k nastavením a možnost návratu zpět.
2. *Posoudit plynulost ovládání a reakci na vstupy:* Zjistit, zda jsou ovládání hry responsivní a plynulé na různých typech zařízení (mobilní telefony a počítače).
3. *Vyhodnotit obtížnost a návratnost hry:* Posoudit, zda jsou první levely hry vhodně vyvážené v obtížnosti a zda hráči mají jasnou představu o tom, co mají udělat a jak hrát.

Dalším důležitým aspektem je vytvoření Výzkumného plánu. Ten zajistí jak výběr správných účastníků, provedení testů a zaznamenání výsledků. Tak i analýzu výsledků a náměty pro zlepšení do příštího kola. Výzkumný plán:

1. *Výběr účastníků:* Pozvat 6 až 12 účastníků, kteří budou postupně po jednom hrát hru.
   (a) Zprvu je výhodnější pozvat hráče, kteří mají zkušenosti s daným žánrem hry.
   (b) Věková skupina by též měla odpovídat zacílení (15 až 35 let).
   (c) Hráči musí mít zkušenost s hraním na mobilních telefonech i PC (Ovlivní to výsledky, pokud hráč nebude mít zkušenost s danou platformou).
2. *Provedení testu:* Účastníci budou hrát po dobu cca 30-40 minut hru a následně odpovídat na otázky moderátora.
   (a) Účastníci musí přijít do neutrálního prostředí (herní studio může ovlivnit objektivitu účastníka).
   (b) Účastníci budou nahráváni zepředu a bude jim nahrávána obrazovka během hraní (tím se zajistí záznam i na pozdější analýzu).
   (c) V místnosti se nachází pouze jeden účastník a moderátor, který dohlíží na správné provedení testu a případné otázky a odpovědi.

(d) V další místnosti je vývojář (případně designer), který si dělá poznámky a má možnost vidět jak na displej, tak na hráče zepředu (Vývojář je oddělen, aby nemohl ovlivnit účastníka).

3. *Zaznamenání výsledků:* Vývojář i moderátor si bude během testování psát poznámky. Moderátor poznamená jak kvantitativní (např. doba strávená v menu, počet pokusů na prvním levelu), tak kvalitativní (např. komentáře hráčů, reakce na ovládání) údaje.

4. *Analýza výsledků:* Získané údaje budou pečlivě analyzovány s důrazem na identifikaci klíčových oblastí pro zlepšení hry, jako jsou problémy s ovládáním, nejasnosti v menu nebo přílišná obtížnost levelů.

5. *Doporučení pro vylepšení:* Na základě analýzy výsledků budou vypracována doporučení pro vylepšení designu a uživatelského zážitku hry.

8.2.2 Výběr vhodných hráčů

Základní aspekty byly popsány ve Výzkumném plánu předchozí kapitoly. Výběr dobrých skupin je důležitý pro získání správného feedbacku. Do výběru by nikdy neměli být zahrnuti lidi, které vývojář nebo firma zná (rodina, přátelé atp.). Tito lidé mají tendenci uměle zveličovat úspěchy a neposkytnou nikdy spolehlivé informace. Seznam předpokladů pro výběr účastníků:

1. *Různorodost zkušeností s hrami:* Účastníci budou vybráni tak, aby zahrnovali širokou škálu zkušeností s hraním her, od nováčků po zkušené hráče. To umožní získat různé perspektivy na uživatelský zážitek.

2. *Různorodost platformy:* Protože hra bude dostupná jak na mobilních zařízeních, tak na PC, účastníci budou vybráni tak, aby zahrnovali uživatele obou platforem. To umožní získat názory a zpětnou vazbu týkající se uživatelského rozhraní a ovládání na obou platformách.

3. *Rozmanitost věkových skupin:* Účastníci budou vybíráni tak, aby zahrnovali různé věkové skupiny, od teenagerů po dospělé. To umožní získat vhled do toho, jak různé věkové skupiny vnímají a interagují s hrou.

4. *Různorodost preferencí ohledně herních žánrů:* Účastníci budou vybíráni tak, aby zahrnovali hráče s různými preferencemi ohledně herních žánrů. To umožní získat poznatky o tom, jak různé typy hráčů vnímají a hodnotí 2D skákačky ve vesmíru ve srovnání s jejich obvyklými preferencemi.

8.2.3 Průběh testování

Samotné testování probíhá za přítomnosti moderátora v místnosti, kde je účastník pouze s ním. Účastník je pozván k hrání hry a požádán, aby vyjádřil své myšlenky a reakce během hraní. Současně je nahráván zepředu a je zachycována obrazovka jeho zařízení. Moderátor sleduje chování účastníka a vede sezení, aby shromáždil relevantní data a odpovědi na otázky. Vývojář pozoruje průběh testování z jiné místnosti a dělá si poznámky na základě pozorovaného chování účastníka. Po skončení hraní moderátor položí následující otázky účastníkovi:

1. Jak jste vnímali ovládání a pohyb postavy ve hře?
2. Jak obtížné bylo pro vás ovládat skákání postavy v různých situacích ve hře?
3. Jak jste vnímali tempo hry a její obtížnost?
4. Jaká byla Vaše reakce na design úrovní a rozložení překážek?
5. Jak jste se cítili ohledně reakcí hry na vaše akce? Měl jste pocit, že hra reagovala dostatečně rychle?
6. Jaký dojem jste měli z vizuálního stylu hry a použitých barev?
7. Jak jste se cítil ohledně hudby a zvuků ve hře? Ovlivnily vaši hratelnost?
8. Jaká byla Vaše zkušenost s obtížností úrovní? Měl jste pocit, že úrovně jsou vyvážené?
9. Jaké byly Vaše reakce na přítomnost nápověd a tutoriálů ve hře? Pomohly vám s orientací?
10. Pocítili jste potřebu pokračovat ve hraní a dosahovat lepších výsledků?

8.2.4 Analýza a zpracování výsledků

Po ukončení všech účastníků moderátor a vývojář projdou veškeré své poznámky a roztřídí je do příslušných částí (např. Herní design, optimalizace, ovládání atp.). Doporučuje se sepsat poznatky na papírky a ty nalepit do sloupečků reprezentující jednotlivé části. Vývojář poté určí konkrétní kroky, které je potřeba podniknout na základě zjištění z testování. Zjištěné problémy a zlepšení jsou implementovány do hry a testování je opakováno k ověření účinnosti provedených změn. Všechny záznamy z testování se archivují pro budoucí použití a referenci. Naplánuje se další řada testů a pozvou se noví účastníci.

9 ANALÝZA ŘEŠENÍ

V této poslední kapitole se práce zaměří na analýzu použitých testovacích metodik a jejich výsledků. Pro test se použila ve většině případů rozpracovaná hra autora. Jedná se o 2D plošinovou skákačku, která se odehrává ve vesmíru a hráč se snaží získat mince. Na hrací ploše je skoro všude tma a hráč si tedy musí svítit malou svítilnou. Hra je plně ovládána pomocí klávesnice a myši. Ve dvou specifických situacích se ale použila hra SidMeier's Civilization V a její nastavení interface. Bylo to z toho důvodu, že se jednalo o testy, kde bylo potřeba mít více aktivních prvků, které mohly být nastaveny v různých kombinacích.

9.1 Kombinatorické testování

Jedná se o dobrý způsob, kdy je potřeba otestovat velké množství funkcí při malém množství testovacích sad. V použitém příkladu je využito homogenního kombinatorického testování, které se právě využívá pro testování herních nastavení nebo HW konfigurace. Jak bylo zmíněno výše, jedná se o jeden ze dvou případů, kdy bylo potřeba využít nastavení ve hře SidMeier's Civilization V. Hra autora nebyla v takové fázi, aby test mohl být prakticky ukázán.

Na konci je vytvořena tabulka, která obsahuje 9 testovacích scénářů, které mohou být použity pro testování. Na Obrázku 6.1 je zobrazeno menu se všemi 13 hodnotami (Spoken Language obsahuje pouze jazyk English), které test tvoří. Z těchto 13 hodnot je 9 typu TRUE/FALSE, dvě hodnoty jsou číselné inputy s možnostmi 0 až 999 a poslední dvě jsou slidery, které nabývají hodnot 0 až 2. Po dokončení je finální tabulka vyobrazena na čtyřech menších Tabulkách 6.1 až 6.4.

Ve výsledku patří mezi jasné výhody efektivita, optimalizace času/zdrojů a pokrytí scénářů. V případě tvorby tabulky mediorem či senior testerem je potřebný čas dobře vynahrazen komplexností tvorby. Po vytvoření se můžou navíc předat hotové testovací scénáře junior testerům a ti budou mít samotné testovaní rychleji a spolehlivost bude dostatečná k vynaloženým zdrojům. Pokrytů sice nebude 100 %, ale mělo by odhalit většinu problému zavčasu.

Na druhou stranu nevýhody jsou v časové náročnosti a složitosti tvorby kombinací. V tomto případě trvalo vytvoření testu cca 3 až 4 hodiny a autor musel několikrát tabulku přepisovat. U 13 sloupů bylo snazší se i přehlédnout a napsat hodnotu, která byla myšlena na jiný řádek. Pokud by nastala změna v parametrech, tabulka by se musela celá předělávat. Tvorbu tabulky by autor nedoporučoval juniorním testerům, kteří neznají systém nebo jen nemají zkušenosti. V tomto případě by tvorba mohla stát i dvojnásobek času.

Kombinatorické testování se ukázalo jako efektivní a účinný způsob testování, zejména při potřebě otestovat velké množství funkcí s minimálním počtem testovacích sad. Jeho výhody zahrnují optimalizaci času a zdrojů, pokrytí různých scénářů a možnost delegovat testování juniorům. Nicméně, jeho implementace může být časově náročná a náchylná k chybám, zejména při tvorbě kombinací a tabulek. Autor doporučuje, aby tvorbu takových testů prováděli zkušenější testeři, aby se minimalizovaly možné chyby a zbytečné ztráty času a peněz. Metodika se méně vyplací v situacích, kdy se má otestovat nízký počet atributů.

9.2 Testovací vývojové diagramy

TVD jsou grafické znázornění modelů, které reprezentují chování hráče. V rámci TVD je vytvořen grafický model, poté následuje datový slovník a na závěr se vytvoří cesty. Vytvořené cesty mohou mít různé kombinace a neplatí názor, že jeden způsob je ten správný. V tomto případě se vytvářely TVD pro pohyb, smrt a sběr mincí v autorově vlastní hře.

Pro pohyb se vytvořil Diagram 6.5. Pokračoval datový slovník a cesty, které byly sepsány. Z cest se vytvořily čtyři testovací scénáře. Dále se vytvořil diagram pro sběr smrt hráče který je reprezentován na Obrázku 6.6. Pro sběr mincí je diagram na Obrázku 6.7. Těmto diagramům byly též vytvořeny datové slovníky a cesty, které vyústily v nové testovací scénáře.

Mezi výhody patří modularita, grafické znázornění a strukturovaný přístup. Díky grafickému znázornění a možnosti flexibilního skládání bloků je možné vytvořit diagramy, které mohou vizuálně testera navádět na testování případů, které jsou více i méně nápadné. Strukturovaný přístup díky datovému slovníku a cestám vytváří dobře napsané testovací scénáře.

Nevýhodami může být složitost, náchylnost na chybou interpretaci a časová náročnost. V případě více komplexního herního mechanizmu může být komplikované vytvoření samotného diagramu, který by byl více nepřehledný. Vytvoření diagramu může být také náchylnější v případě chybné interpretace. Opět se zde vyplácí, aby tvorbu těžších TVD prováděl zkušenější tester a jeho aplikaci pak může provádět i junior. Tím se odvíjí i časová náročnost, která je stejně jako u kombinatorických testů, ale zde spíše způsobena tvorbou více částí.

TVD a jejich modulární povaha umožňuje flexibilní skládání diagramů, což usnadňuje tvorbu testovacích scénářů a vizuálně navádí testery k testování různých případů užití. Strukturovaný přístup s datovými slovníky a cestami pak přispívá k vytváření jasných a dobře napsaných testovacích scénářů. Nicméně, složitost tvorby TVD a náchylnost k chybám při interpretaci mohou být výzvou, zejména pro komplexní herní

mechanismy. Metoda je vhodná pro test menších i středních velikostí herních mechanismů. Stejně jako v předchozím případě je lepší, když tvorbu složitějších TVD provádějí zkušení testeři.

9.3 Cleanroom testování

Tento způsob testování je založen na předpokladu, že testování je prováděno stejně, jako by hru hráli hráči. Využívá pravděpodobnost použití, která se může zjistit jak analytickými nástroji, dotazníky, očekáváním týmu nebo např. open beta testováním. Jedná se o druhý a poslední test, který byl prováděn na hře SidMeier's Civilization V. Využívá totiž již existující kombinatorickou tabulku, která se pouze upraví.

Tabulky 6.5, 6.7, 6.9 a 6.11 reprezentují tabulku s pravděpodobností pro hráče Achiever. Po náhodně vygenerovaných hodnotách se vytvořily Tabulky 6.6, 6.8, 6.10 a 6.12, které reprezentují finální tabulku s devíti testovacími scénáři, které pro tohoto hráče mohou nejčastěji nastat.

Mezi výhody zde patří stejně jako v případě kombinatorických efektivita, optimalizace času/zdrojů a pokrytí scénářů. Pro nevýhody to platí stejně tak, jsou jimi časové náročnosti a složitosti tvorby kombinací. Ovšem v čem se liší je jejich soustředění na různé typy hráčů, režimu nebo reálného života. Pokud má tým k dispozici kvalitní data o hráčích, lze díly těmto testům vyladit hromadu herních metodik přímo pro hráče. Tato metodika je vhodná do všech týmů, kde jsou schopni využít nasbíraná data.

9.4 Funkcionální testování

Je to nejběžnější způsob testu, kdy je cílem jednoduše ověřit, že požadovaná funkcionalita funguje. V práci se takto zaměřilo přímo na chování všech komponent, které jsou k dispozici během hraní. Přesněji se testoval pohyb kosmonauta, sběr mincí a jejich aktualizace v infopanelu, osvětlení, zkouška jednotlivých pastí, ubíhající časomíra a obecná hratelnost.

Je brán jako jeden ze základních pilířů testování jakéhokoliv SW. Může pomoci jak s testováním přímo na reálném prostředí, tak zároveň prohloubit zkušenosti s aplikací pro juniornější testery. Zároveň je ale potřeba mít další metody pro testování (např. výkonnostní nebo unit testování), protože funkcionální testování se soustředí jen na správnost funkčnosti SW. Také může být nákladné, pokud má tester projít více procesů manuálně. Ve výsledku by toto testování mělo (často i je, aniž by o tom vědělo) být použito ve všech týmech, kdy ani nezáleží na velikosti týmu nebo aplikace.

9.5 Unit testování

Jeden z dalších testovacích způsobů je Unit testování. To se zaměřuje na testování jednotlivých komponent, tříd nebo metod v SW aplikace. Kvůli vyvíjení v nástroji Unity jsou testy prováděny přes Test Framework a spouštěny přes Test Runner. Zde jsou rozděleny na EditMode a PlayMode, kde rozdíl tkví, že PlayMode testy jsou spuštěny se hrou, zatímco EditMode ne. V rámci ukázky se testovaly třídy *Timer* a *EnemyMovement*.

Pro každou z nich byl vytvořen vlastní testovací soubor, který obsahoval sadu testů. Přesněji pro každou třídu vznikly 4 testy a ke konci každé sady byl doložen screenshot z Code coverage, kde se mapovalo pokrytí. Ty jsou viditelné na Obrázcích 7.2 a 7.4.

K výhodám patří zajištění okamžitého feedbacku vývojáři, který po vytvoření nové funkcionality spustí testy, aby zjistil, jestli nic nepokazil a jestli se funkcionalita chová správně. Dále dobrá podpora po refaktorizaci, kdy po změnách opět může vniknout rychlá kontrola, že se nic nezměnilo, v rámci logiky. Samotný test je daleko rychlejší, než pokud by se měla stejná logika testovat manuálně.

Naopak jako nevýhoda je následná údržba testů, kdy pokud se změní logika, musí být testy znovu vytvořeny a otestovány. Pořád mají omezené pokrytí a jejich význam by měl být pouze v testování jednodušších prvků v kódu (mohou být náročné pro testování interakcí mezi různými jednotkami kódu). Některé testy mohou být falešně pozitivní, pokud jsou napsány nevhodně nebo nedostatečně.

Opět je brán jako jeden z pilířů testování SW a měl by být využíván pro jakýkoliv tým nehledě na velikost týmu nebo aplikace samotné. Kvalita testů je úměrná kvalitě psaní tříd. Pokud splňují podmínky SOLID a OOP, tak jsou testy kvalitnější a více stabilní. Zároveň pokud se myslí na omezené krytí, pravidelnou údržbu, která je důležitá, aby vše fungovalo správně a náklady, aby nebyly zbytečně vysoké. A dostatečně se kontroluje tvorba testů, aby byly napsány logicky správně, tak je to dobrý nástroj pro testování základních komponent v SW. Jeho výhody jako okamžitý feedback, využití při refaktorizaci kódu a rychlost samotného spuštění pak jen podtrhávají důležitost tohoto typu testování.

9.6 Výkonnostní testování

Pro změření stability a robustnosti systému ve specifických situacích se testeři zaměří na Výkonnostní testování. Unity nemá přímo nástroje pro testování výkonu, existuje však několik způsobů, jak toto testovat. Nejčastěji si vývojáři napíší vlastní zátěžové testy stejně jako Unit testy přes Test Framework a následně je spouští přes Test Runner. Případně si vytvoří testovací verze, které dají do určitého HW (mobilní telefon, konzole atp.) na kterém testují optimalizaci. Do toho využívají nástroj Profiler, který dovede

měřit výkonnost jednotlivých HW komponent jako využití CPU, GPU nebo paměti. V kapitole o výkonnostním testování je k vidění Obrázek 8.1 a metody pro testování FPS a rychlosti načítání scén.

Mezi největší výhody patří optimalizace SW, aby bylo možné dopřát hráči co nejlepší zážitek bez ohledu na jeho hardware. S tímto se pojí i vyšší prodeje, které jsou spojené s vyšší spokojeností hráčů. Naopak k nevýhodám je přisuzována časová náročnost testů, která může být delší pro vytvořené správných podmínek. Dále pak vyšší náklady spojené s HW, který je potřeba pořídit pro optimalizaci a případná koupě SW pro lepší analýzu.

V souhrnu je dobré počítat s výkonnostními testy ať už u AAA her nebo u malých indie hříček. V případě velkých her by se mělo jednat o samozřejmost, že hra poběží na poskytnutém seznamu HW optimalizovaně a nebudou žádné propady snímků nebo dlouhé načítání scén. Pro indie vývojáře je dobré mít otestovanou jejich hru alespoň na nejslabších zařízeních, na kterém má být, aby hra nebyla tzv. nehratelná.

9.7 Testování použitelnosti

Je to cenný nástroj pro získání zpětné vazby od skutečných hráčů, kteří budou hru hrát. Od zbytku metod, které jsou v této kapitole zmíněny je odlišná v tom, že pracuje s obyčejnými hráči, kteří nejsou jinak zaujatí hrou tím, že na ní pracovali. Pro uskutečnění je potřeba si nachystat cíle, které se mají splnit, nechat otestovat hru hráči a zpracovat výsledky. V praktické části v kapitole o Testování použitelnosti je celý postup toho, jak postupovat a jak se snažit získat co nejvíce užitečných informací.

K výhodám patří zlepšení herního zážitku pro hráče, který je schopný ve hře strávit více času. Dále snížení frustrace hráče, což může vyústit ke zvýšená loajality a tím pádem i možným budoucím ziskům. A samotný lepší komerční úspěch, bez kterého by se nemusel vývoj zaplatit.

Je ale potřeba poznamenat, že tato metoda (obzvlášť pokud je aplikována několikrát) je dosti nákladná (potřeba sehnat místo, účastníky, moderátora) a časově náročná. Zároveň je potřeba mít člověka, co dovede být dobrý moderátor a dokáže z hráčů vytáhnout co nejvíce užitečných informací. Pokud se jedná o očekávaný titul, je možné i vyšší riziko spoilerů.

Jedná se o metodiku, která se více vyplácí spíše větším týmům, které si ji mohou dovolil, ale její výhody v případě úspěšného aplikování mohou převýšit finanční i časové nevýhody. Teoreticky lze využít u menších her možnost demo verze, nebo možné uzavřené testování pro veřejnost. Naopak se ale musí myslet na možnosti spoilerů nebo únik informací, či špatnou interpretaci dat.

ZÁVĚR

Diplomová práce se zabývala prozkoumáním a analýzou testování v herním průmyslu. Dalším cílem bylo v poskytnutí užitečných poznatků a doporučení pro vývojářské týmy, které by jim mohly pomoci s vývojem, a hlavně testováním jejich her. V poslední řadě šlo i o autorovu snahu lépe porozumět těmto procesům a prohloubit si jeho znalosti o testování softwaru ve hrách.

Práce začala představením stručné historie, kde byly krátce popsaly začátky testování od roku 1958 až po současnost. Následovala obecná rovina testování SW. Byly popsaly základní typy testování (manuální a automatizované) od kterých se rozvíjely další druhy jako Funkcionální, Nefunkcionální a ostatní druhy. Funkcionální testování se dále rozděluje na Unit testování, Integrační testy, Systémové testování a Akceptační testování. U Nefunkcionálního testování to je Bezpečnostní testování, Výkonnostní testování a Testování použitelnosti. Samotné oblasti mají ještě i své podoblasti, ale ty už nebyly popsány, protože téma práce bylo dostatečně obsáhlé a autor usoudil, že podoblasti by neměly takový přínos. Následovala kapitola o Herním průmyslu, kde byly rozebírány jednotlivé herní platformy (plus jejich zhodnocení) a samotný vývoj v herním průmyslu. Po herním průmyslu následovala kapitola o herním testování. V této kapitole se přiblížily cykly herního vývoje, typy defektů a samotné testovací techniky. Ke každé z technik byl přidán příklad pro názornou ukázku.

V praktické části nejdříve došlo ke kontaktování dvou herních studií a položením čtrnácti otázek ohledně jejich vývoje a testování. Tímto byla možnost nahlédnout pod pokličku herním studiím a získat tak pohled na testování v praxi. Následovala kapitola s testovacími metodami, které byly aplikovány na testování autorovy hry, která je ve vývoji. Ovšem v případě Kombinatorického testování a Cleanroom testování byla použita již existující hra Sid Meier's Civilization V. Bylo to z důvodu nedostatečné komplexnosti autorovi hry, kde nebylo možné danou metodu kvalitně použít. Kombinatorická tabulka reprezentovala možnosti účinného testu komplexních systémů za pomoci malého množství testovacích sad. Cleanroom vycházel z této tabulky a využil pro otestování pravděpodobnosti užívání u daných hráčů, čímž zároveň hru optimalizuje pro cílovou skupinu. Testovací vývojové diagramy sloužily k testování různých systémů a odhalení méně častých testovacích scénářů. K nim se pak sepsaly datové sklady s popisem událostí a vytvoření cest, ze kterých vznikly testovací scénáře. V rámci funkcionálního testování byly vytvořeny základní testovací scénáře pokrývající základní prvky ve hře jako hráč, osvětlení, nebo časomíra. U Unit testování byly popsány dvě testovací sety (soubory), které sloužily pro test třídy Timer a Enemy_movement. Dále se u výkonnostního testování byly opět popsaly napsané testy na kontrolu FPS a délky načítání scén. Jako poslední metoda byla použita metoda Funkcionálního testování, která

vytvořila přípravu pro otestování hry účastníky, kteří následně dají zpětnou vazbu. V poslední řadě byly analyzovány výsledky testování a krátce se popsalo, pro které týmy je vhodné dané metody použít.

Herní průmysl je stejně jako jakékoliv jiné odvětví dost rozdílně od zbytky a má svá specifika, ale v rámci testování a metod, které byly použity, tak autor nevidí velký rozdíl oproti jiným odvětvím. Jistě dílčí elementy se mohou lišit, ale ve zkratce jakákoliv metoda zde použita může být využita i například ve finančním sektoru nebo e-shopu. Zajímavý by mohl být budoucí vývoj testování díky nedávnému pokroku v AI. Již teď je možné psát testovací scénáře pouze se zadáním dobrého promptu anebo si pomoct se psaním automatizovaných testů díky AI (např. v Postmanu). Pokud je tester zdatný, může již teď zvládnout otestovat více úkolu za nižší jednotku času a zbylý čas investovat do automatizace, která pomůže jemu i týmu v budoucím vývoji. Ovšem cesta k plnému nahrazení testeru umělou inteligencí je zatím dlouhá a strastiplná, tudíž o zaměstnání se nikdo bát nemusí.

SEZNAM POUŽITÉ LITERATURY

[1] PATTON, Ron. Testování softwaru. Praha: Computer Press, 2002. ISBN 80-7226-636-5.

[2] ISTQB [online]. Brusel: ISTQB not-for-profit association, 2022 [cit. 2022-11-06]. Dostupné z: https://www.istqb.org.

[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.

[6] A brief history of software testing [online]. 2019 [cit. 2024-01-04]. Dostupné z: https://salsa.digital/insights/a-brief-history-of-software-testing.

[7] What is software testing? Online. 2024. Dostupné z: https://www.ibm.com/topics/software-testing. [cit. 2024-01-04].

[8] History of Software Testing [online]. 2022 [cit. 2024-01-04]. Dostupné z: https://www.geeksforgeeks.org/history-of-software-testing/.

[9] The different types of software testing [online]. 2024 [cit. 2024-01-04]. Dostupné z: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

[10] Unit Testing [online]. 2024 [cit. 2024-01-04]. Dostupné z: https://tuskr.app/learn/unit-testing.

[11] What is Integration Testing: Examples, Challenges, and Approaches [online]. 2022 [cit. 2024-01-04]. Dostupné z: https://www.simplilearn.com/what-is-integration-testing-examples-challenges-approaches-article

[12] What is Functional Testing? Definition, Types & Examples [online]. 2023 [cit. 2024-01-04]. Dostupné z: https://katalon.com/resources-center/blog/functional-testing

[13] What is Non-Functional Testing? [online]. 2023 [cit. 2024-01-05]. Dostupné z: https://www.browserstack.com/guide/what-is-non-functional-testing

[14] What is Acceptance Testing? (Importance, Types & Best Practices) [online]. 2022 [cit. 2024-01-04]. Dostupné z: https://www.browserstack.com/guide/acceptance-testing

[15] What Is Security Testing? [online]. 2023 [cit. 2024-01-05]. Dostupné z: https://www.hackerone.com/knowledge-center/what-security-testing

[16] Performance testing, best practices, metrics & more [online]. 2023 [cit. 2024-01-07]. Dostupné z: https://www.tricentis.com/learn/performance-testing

[17] A Beginner's Guide to Usability Testing [online]. 2024 [cit. 2024-01-07]. Dostupné z: https://maze.co/guides/usability-testing/

[18] R. S. Pressman, Software Engineering: A Practioner Approach, 5th ed. (Book style), New York City: John Wiley & Sons, 2001.

[19] Gaming Industry Overview [online]. 2022 [cit. 2024-01-07]. Dostupné z: https://builtin.com/gaming

[20] PC and PS5 are Developers Favorite Platforms, Stadia and Similar Services Get Little Love [online]. 2022 [cit. 2024-01-07]. Dostupné z: https://wccftech.com/pc-ps5-developers-favorite-platforms-gdc-poll/

[21] Gaming Platforms: A Comprehensive Guide [online]. 2023 [cit. 2024-01-07]. Dostupné z: https://medium.com/@charleskexley/gaming-platforms-a-comprehensive-guide-40b1e3944220

[22] What is cloud gaming? [online]. 2021 [cit. 2024-01-07]. Dostupné z: https://www.digitaltrends.com/gaming/what-is-cloud-gaming-explained/

[23] How AR & VR in Gaming is Transforming Gameplay Experience? [online]. 2023 [cit. 2024-01-07]. Dostupné z: https://300mind.studio/blog/ar-vr-in-gaming-industry/

[24] Agile vs Waterfall Localization in Game Development: Pros, Cons, and How to Do It Right [online]. 2020 [cit. 2024-01-07]. Dostupné z: https://www.gridly.com/blog/agile-localization/

[25] Game Development: Project Lifecycle [online]. 2020 [cit. 2024-02-17]. Dostupné z: https://web.archive.org/web/20200219170413/http://www.blitzgamesstudios.com/blitzacad

[26] Project Management for Game Development [online]. 2009 [cit. 2024-02-17]. Dostupné z: https://web.archive.org/web/20201127150213/http://mmotidbits.com/2009/06/15/project-management-for-game-development/

[27] The Game Development Lifecycle - A theory for the extension of the Agile project methodology [online]. 2011 [cit. 2024-02-17]. Dostupné z: https://web.archive.org/web/20131230054620/http://blog.dopplerinteractive.com/2011/04/g development-lifecycle-theory-for.html

[28] 20 Types of Software Defects Every Tester Should Know. Online. Software testing material. 2024. Dostupné z: https://www.softwaretestingmaterial.com/types-of-software-defects/?utm_source=rss&utm_medium=rss&utm_campaign=types-of-software-defects. [cit. 2024-02-28].

[29] Combinatorial Testing | What it is, How to Perform & Tools. Online. Testsigma. 2023. Dostupné z: https://testsigma.com/blog/combinatorial-testing/. [cit. 2024-03-02].

[30] Test Flow Diagram – A Test Graphing Technique. Online. Rishabh software. 2012. Dostupné z: https://www.rishabhsoft.com/blog/test-flow-diagram-a-test-graphing-technique. [cit. 2024-03-02].

[31] Cleanroom Software Engineering Reference Model. Online. Rishabh software. 1996. Dostupné z: https://insights.sei.cmu.edu/documents/1159/1996_005_001_16502.pdf. [cit. 2024-03-04].

[32] Tree Testing: Fast, Iterative Evaluation of Menu Labels and Categories. Online. Nielsen Norman Group. 2023. Dostupné z: https://www.nngroup.com/articles/tree-testing/. [cit. 2024-03-04].

[33] Adhoc Testing in Software. Online. Geeks for Geeks. 2024. Dostupné z: https://www.geeksforgeeks.org/adhoc-testing-in-software/. [cit. 2024-03-04].

[34] How unity 1 looked back in 2005. Online. 2020. Dostupné z: https://forum.unity.com/threads/how-unity-1-looked-back-in-2005.148312/. [cit. 2024-02-17].

[35] Unity: Development History and the Influence of This Game Engine on the Game Development Industry. Online. 2023. Dostupné z: https://medium.com/wotammorpg/unity-development-history-and-the-influence-of-this-game-engine-on-the-game-development-36dc7a7a3b9d. [cit. 2024-02-17].

[36] Unity 2.0 game engine now available. Online. 2007. Dostupné z: https://www.macworld.com/article/187693/unity-18.html. [cit. 2024-02-17].

[37] Unity 3 brings very expensive dev tools at a very low price. Online. 2010. Dostupné z: https://arstechnica.com/information-technology/2010/09/unity-3-brings-very-expensive-dev-tools-at-a-very-low-price. [cit. 2024-02-17].

[38] Unity 4.0 available for download today with DX 11 support and Linux preview. Online. 2012. Dostupné z: https://www.polygon.com/2012/11/14/3645122/unity-4-0-available-download. [cit. 2024-02-17].

[39] Unity 5 Announced With Better Lighting, Better Audio, And "Early" Support For Plugin-Free Browser Games. Online. 2014. Dostupné z: https://techcrunch.com/2014/03/18/unity-5-announced-with-early-support-for-plugin-free-browser-games/. [cit. 2024-02-17].

[40] The road to 2021. Online. 2020. Dostupné z: https://blog.unity.com/technology/the-road-to-2021. [cit. 2024-02-17].

[41] Unity MARS Augmented and Mixed Reality authoring studio now available. Online. 2020. Dostupné z: https://www.auganix.org/unity-mars-augmented-and-mixed-reality-authoring-studio-now-available/. [cit. 2024-02-17].

[42] Tech Stream release Unity 2022.1. Online. 2022. Dostupné z: https://unity.com/releases/2022.1#enhanced-productivity. [cit. 2024-02-17].

[43] The next version of Unity will be called Unity 6. Online. 2023. Dostupné z: https://www.gamedeveloper.com/business/the-next-version-of-unity-will-be-called-unity-6. [cit. 2024-02-17].

[44] ESPN NFL 2k5 Sliders - User vs CPU Simulation Settings [online]. 2020 [cit. 2024-02-23]. Dostupné z: https://www.youtube.com/watch?v=dJdsGOGF33M

[45] Black Pendant. Yugipedia [online]. 2023 [cit. 2024-03-17]. Dostupné z: https://yugipedia.com/wiki/Black_Pendant

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

CPU	Central Processing Unit
TDD	Test-driven development
BDD	Behavior-driven development
API	Application Programming Interface
AI	Artificial Intelligence
QA	Quality assurance
SW	Software
HW	Hardware
OS	Operating System
GUI	Graphical User Interface
SQL	Structured Query Language
VR	Virtual reality
AR	Augmented reality
GDSE	Game Development Software Engineering
XP	Extreme Programming
SE	Software Engineering
SDLC	Software development lifecycle
GDLC	Game development lifecycle
IDE	integrated development environment
RPG	Role-playing game
MMO	Massively Multiplayer Online game
ODC	The orthogonal defect classification
MMORPG	Massively Multiplayer Online Role-playing game
TVD	Testovací vývojové diagramy
MTTF	mean time to failure
RTS	Real-time strategy
LAN	Local Area Network
CFO	Chief Financial Officer
FPS	Frames per second

SEZNAM OBRÁZKŮ

Obr. 2.1. Rozdělení testovacích typů ... 14
Obr. 3.1. Waterfall metoda s lokalizací ... 28
Obr. 3.2. Agile metoda s lokalizací ... 28
Obr. 4.1. Blitz game development life cycle ... 31
Obr. 4.2. Arnold Hendrick's GDLC ... 31
Obr. 4.3. GDLC s projektovým životním cyklem ... 33
Obr. 4.4. Doppler Interactive GDLC ... 33
Obr. 4.5. Tři diagramy charakterující různé variace Stromu verzí ... 38
Obr. 4.6. ESPN NFL 2K5 Nastavení[44] ... 45
Obr. 4.7. Flow komponenty ... 50
Obr. 4.8. TVD diagram I ... 52
Obr. 4.9. TVD diagram II ... 53
Obr. 4.10. TVD diagram III ... 54
Obr. 4.11. Příklady tabulek ke Cleanroom testování ... 64
Obr. 4.12. Strom testovacích případů I ... 68
Obr. 4.13. Strom testovacích případů II ... 69
Obr. 4.14. Technologický strom pro Eldar Aspect Portal ... 69
Obr. 4.15. Stromy povolání modrého mága a iluzionisty pro lidi ve hře FFTA ... 70
Obr. 6.1. Civilization V Interface menu ... 81
Obr. 6.2. Ukázka vyvíjené hry ve světle ... 86
Obr. 6.3. První část vývojového diagramu pro pohyb ... 87
Obr. 6.4. Druhá část vývojového diagramu pro pohyb ... 87
Obr. 6.5. Finální vývojový diagramu pro pohyb ... 88
Obr. 6.6. Vývojový diagramu pro smrt hráče ... 90
Obr. 6.7. Vývojový diagramu pro mince ... 93
Obr. 7.1. Ukázka vyvíjené hry ... 99
Obr. 7.2. Code coverage pro třídu Timer ... 105
Obr. 7.3. Code coverage pro třídu Timer ... 105
Obr. 7.4. Code coverage pro třídu Enemy_movement ... 107
Obr. 8.1. Unity Profiler ... 111

SEZNAM TABULEK

Tab. 1.1. Historické milníky testování ... 13
Tab. 4.1. Testovací tabulka I ... 45
Tab. 4.2. Testovací tabulka II ... 45
Tab. 4.3. Testovací tabulka III ... 46
Tab. 4.4. Testovací tabulka IV ... 46
Tab. 4.5. Testovací tabulka V ... 47
Tab. 4.6. Testovací tabulka VI ... 48
Tab. 4.7. Testovací tabulka VII ... 48
Tab. 4.8. Souhrn základní cesty ... 58
Tab. 4.9. Souhrn odborné cesty ... 59
Tab. 4.10. Výběr metodiky návrhu testů ... 61
Tab. 4.11. Hotová Cleanroom kombinatorická tabulka pro příležitostného hráče ... 65
Tab. 4.12. Hotová Invertovaná tabulka pro Look sensibility ... 67
Tab. 4.13. Testy vlastností stromu pro lidského modrého mága ... 70
Tab. 4.14. Testy vlastností stromu pro lidského iluzionistu ... 70
Tab. 6.1. Kombinatorická tabulka pro Interface settings v Civilization V (část I.) 83
Tab. 6.2. Kombinatorická tabulka pro Interface settings v Civilization V (část II.) ... 84
Tab. 6.3. Kombinatorická tabulka pro Interface settings v Civilization V (část III.) ... 85
Tab. 6.4. Kombinatorická tabulka pro Interface settings v Civilization V (část IV.) ... 85
Tab. 6.5. Tabulka s hodnotami pro hráče zaměřeného na úspěchy (Achievera) ... 94
Tab. 6.6. Cleanroom kombinatorická tabulka pro Interface settings v Civilization V (část I.) ... 95
Tab. 6.7. Tabulka s hodnotami pro hráče zaměřeného na úspěchy (Achievera) ... 95
Tab. 6.8. Cleanroom kombinatorická tabulka pro Interface settings v Civilization V (část II.) ... 96
Tab. 6.9. Tabulka s hodnotami pro hráče zaměřeného na úspěchy (Achievera) ... 96
Tab. 6.10. Cleanroom kombinatorická tabulka pro Interface settings v Civilization V (část III.) ... 97
Tab. 6.11. Tabulka s hodnotami pro hráče zaměřeného na úspěchy (Achievera) ... 97
Tab. 6.12. Cleanroom kombinatorická tabulka pro Interface settings v Civilization V (část IV.) ... 98