УДК 004.89:004.93:004.4'2

АНАЛІЗ ПРОБЛЕМ ТА ІСНУЮЧИХ РІШЕНЬ АВТОМАТИЗАЦІЇ ПОШУКУ ФАЙЛІВ В ІГРОВИХ ЗАСТОСУНКАХ

Козаченко Д. О.[1]; Нестеров Ю. М.[2]
[1,2] Національний університет «Одеська політехніка»
[1,2] Україна, Одеса
[1] 7491631@stud.op.edu.ua; [2] nesterov@op.edu.ua

Анотація. *У тезах проаналізовано технічні проблеми пошуку й вилучення текстових ресурсів, графічних асетів і конфігурацій у скомпільованих контейнерах сучасних відеоігор для потреб локалізації та модингу. Досліджено існуючі засоби розпакування архівів і методи штучного інтелекту для класифікації файлів, обґрунтовано архітектуру комбінованої програмної системи.*
Ключові слова: *локалізація ігор; модинг; аналіз ігрових директорій; ідентифікація файлів; глибинні нейронні мережі.*

Вступ. Локалізація (адаптація та переклад ігор іншими мовами) і модинг (створення користувацьких модифікацій) є важливими процесами для підтримування популярності відеоігор на світовому ринку. В ідеальному випадку розробники зберігають ігровий текст у відкритих таблицях (наприклад, у текстових файлах формату CSV або через інтеграцію з хмарними таблицями), які легко редагувати у табличних процесорах. Проте в більшості сучасних ігор ресурси приховані всередині скомпільованих бінарних контейнерів [1]. Оскільки наявні засоби зворотної розробки є недосконалими (стандартні програми вимагають заголовків файлів, а методи частотного аналізу працюють надто повільно та неточно на великих обсягах даних) [2], фахівцям із локалізації доводиться досліджувати структуру архівів і вилучати тексти чи шрифти повністю вручну [1].

Мета дослідження – систематизація технічних перешкод автоматизованого пошуку ресурсних файлів у ігрових застосунках, критичний аналіз існуючих програмних та інтелектуальних методів ідентифікації даних, а також обґрунтування архітектури комбінованої програмної системи на базі штучного інтелекту (ШІ) для автоматизації локалізації та модингу.

Аналіз основних проблем предметної області. Дослідження структури ігрових застосунків дозволяє виділити чотири головні проблеми, які перешкоджають автоматичному пошуку файлів:

1. Закриті формати ігрових рушіїв та відсутність метаданих. Замість стандартних директорій ігри використовують великі файли-контейнери, де видалені розширення та оригінальні назви файлів [1]. Наприклад, в ігровому рушії Unity ресурси пакуються у бінарні контейнери AssetBundle та SerializedFile. Для їх читання потрібна спеціальна схема даних – «дерево типів» (TypeTree), але у фінальних збірках ігор розробники часто видаляють її для захисту від злому та зменшення обсягу дистрибутива. У рушії Unreal Engine 5 на зміну класичним архівам .pak прийшла система IoStore (Zen Loader), яка агрегує дані у файли .utoc і .ucas та шифрує їх симетричним алгоритмом AES (Advanced Encryption Standard).

2. Використання складних алгоритмів стиснення. Щоб зменшити розмір гри та прискорити її завантаження, компанії застосовують комерційні алгоритми стиснення, зокрема сімейство Oodle Data (алгоритми Kraken, Mermaid, Selkie, Leviathan). Наприклад, алгоритм Kraken стискає дані на 10–15% ефективніше за стандартний ZIP і розпаковує їх учетверо швидше. У стисненому стані такі файли характеризуються максимальною інформаційною ентропією (виглядають як

випадковий набір байтів), тому виявити в них текстові рядки шляхом прямого пошуку символів неможливо без попереднього розпакування даних [1, 2].

3. Змішування тексту діалогів із програмним кодом. Навіть після розпакування архівів репліки персонажів рідко зберігаються окремо від логіки гри. У файлах сценаріїв звичайні репліки персонажів (наприклад, «I hope everyone else») чергуються з технічними командами управління сценою та анімацією: position(player, above), squeak(cry), speak_active, endcutscene(). Якщо програма автоматично експортує всі текстові дані поспіль, перекладач отримає величезний масив програмного коду, а випадкова модифікація системної інструкції призведе до аварійного завершення гри.

4. Нестандартні кодування та графічні шрифти. Значна частина ігор використовує специфічні таблиці символів (наприклад, японське кодування Shift-JIS замість універсального стандарту UTF-8 – Unicode Transformation Format 8-bit) [1]. Помилкове визначення кодування спричиняє появу нечитабельних символів – ефекту mojibake. Крім того, ігри часто візуалізують текст не векторними системними шрифтами, а за допомогою растрових атласів із літерами, які програмній системі необхідно окремо ідентифікувати серед тисяч інших текстур.

Аналіз існуючих рішень та їх обмежень. На сьогодні наявні програмні засоби та методи, які можна застосувати для пошуку й аналізу ресурсів в ігрових директоріях, поділяються на три основні групи:

1. Стандартні програми для розпакування та утиліти під ігрові рушії. Найвідомішим універсальним інструментом є скриптовий розпакувальник QuickBMS, який підтримує сотні форматів і здатний підключати динамічну бібліотеку oo2core_4_win64.dll для декодування алгоритмів Oodle. Для проектів на базі Unity активно використовуються AssetStudio, AssetRipper та бібліотека UnityPy, для Unreal Engine – FModel та UnrealReZen, а для візуальних новел – GARbro, VNTextPatch, GalTransl та спеціалізований інструмент аудиту Galgame Doctor [1]. Обмеження: ці програми функціонують за жорсткими сигнатурними шаблонами – вони шукають розширення файлів або фіксовані «магічні байти» заголовка (наприклад, послідовність FF D8 на початку зображень JPEG або GIF89a у файлах GIF). Якщо розробники модифікували заголовок, видалили TypeTree або оновили версію рушія (зокрема у Unity 2023.2+ та Unity 6), парсери втрачають працездатність.

2. Методи розпізнавання ігрових ресурсів за частотою байтів. Оскільки в архівах ігор часто видалені формати файлів, для їхнього пошуку інколи застосовують підходи з цифрової криміналістики (наприклад, File Carving). Замість пошуку конкретного формату, алгоритм математично аналізує «начинку» невідомого фрагмента гри: обчислює частотний розподіл байтів (від 0 до 255) та вимірює ентропію Шеннона, класифікуючи дані методом опорних векторів (SVM). Типовою програмою цього класу є система Sceadan [2]. Обмеження: цей підхід є занадто повільним і неточним для локалізації сучасних ігор. На масштабному наборі даних (FFT-75) Sceadan правильно розпізнає лише 69% файлів і витрачає близько 9 хвилин на аналіз 1 ГБ інформації [2, 3]. Головна проблема полягає в тому, що з точки зору статистики технічний код гри (скрипти) та звичайний текст реплік персонажів мають майже ідентичні частотні профілі. Через це алгоритм постійно плутає їх, видаючи перекладачу змішаний програмний код замість чистих діалогів.

3. Інструменти на базі штучного інтелекту та нейромереж. Для подолання недоліків частотного аналізу дедалі частіше застосовують згорткові нейромережі (CNN) та алгоритми обробки природної мови (NLP). Наприклад, нейромережевий інструмент FiFTy здатний класифікувати фрагменти файлів із точністю до 77,5%, обробляючи 1 ГБ невідомих даних лише за 38 секунд [3]. Новітні оптимізовані нейромережі працюють у 24 рази швидше та

впізнають понад 90% прихованих графічних і аудіофайлів [4]. Візуальні моделі (Byte2Image, ByteNet) перетворюють байти на зображення, досягаючи 93% точності на мультимедійних ресурсах [5]. А для найскладнішої задачі – відокремлення реплік персонажів від службового коду – найвищу результативність демонструють сучасні мовні моделі (FastText, BERT, DistilBERT), які розпізнають текст із точністю понад 98% [6]. Обмеження: усі ці потужні алгоритми ШІ досі існують лише як окремі наукові розробки для криміналістів. Вони не інтегровані з програмами-розпакувальниками ігор у єдину прикладну систему, якою могли б користуватися перекладачі та модороби. Узагальнене порівняння розглянутих рішень наведено у табл. 1.

Таблиця 1 – Порівняльний аналіз існуючих засобів і методів пошуку та класифікації ігрових ресурсів

Група рішень	Приклади засобів	Головний принцип роботи	Основні недоліки у задачах локалізації та модингу
Стандартні програми-розпакувальники	QuickBMS, AssetStudio, FModel, UnityPy, GARbro [1]	Пошук за розширеннями, магічними байтами та відомими схемами (TypeTree, .utoc)	Не працюють у разі зміни заголовків, шифрування або оновлення версії рушія; не фільтрують діалоги від коду
Методи частотного аналізу байтів	Sceadan, частотний аналіз та ентропія Шеннона [2]	Підрахунок частоти байтів (0–255) та класифікація еталонів через SVM	Повільна робота (9 хв/ГБ), низька точність на великій кількості форматів (69%), плутанина між кодом і текстом
Інструменти на базі ШІ та нейромереж	FiFTy [3], DSC-SE CNN [4], ByteNet [5], FastText, BERT [6]	Глибокий аналіз послідовностей байтів (1D/2D CNN) та семантики рядків (NLP)	Вузька спеціалізація: не здатні самостійно розпаковувати ігрові архіви (Oodle) та експортувати переклад

Висновки. У результаті проведеного дослідження встановлено, що жоден із наявних інструментів окремо не забезпечує повного циклу автоматичного пошуку ресурсів у сучасних ігрових застосунках: стандартні програми-розпакувальники втрачають працездатність через видалення заголовків і комерційні алгоритми стиснення [1], а спеціалізовані моделі ШІ не адаптовані до архітектури ігрових контейнерів [3–5]. Обґрунтовано, що найбільш перспективним рішенням є розроблення комбінованої програмної системи, яка поєднує модуль автоматичної декомпресії ігрових архівів (зокрема Oodle) зі швидкою згортковою нейромережею (на базі FiFTy / DSC-SE CNN) для ідентифікації бінарних асетів без метаданих та двоступеневим каскадом мовних моделей (FastText і DistilBERT) для високоточної фільтрації ігрових діалогів від програмного коду.

Використання інструментів ШІ. Під час підготовки тез доповіді використовувався інструмент штучного інтелекту Google Gemini (модель Gemini 2.5 Pro) з метою систематизації огляду літературних джерел, перевірки коректності термінології та стилістичного редагування тексту в розділах вступу й аналізу існуючих рішень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] JerType. (2023). Game localization and reverse engineering binary formats. *Legends of Localization*. Retrieved from https://www.jertype.com/game-localization/

[2] Karampidis, K., Kavallieratou, E., & Papadourakis, G. (2018). File type identification – Computational intelligence for digital forensics. *Journal of Systems and Information Technology*, 20(1), 4–26. https://doi.org/10.1108/JSIT-06-2017-0047

[3] Mittal, G., Korus, P., & Memon, N. (2019). FiFTy: Large-scale file fragment type identification using neural networks. *arXiv preprint arXiv:1908.06148*, 1–13. Retrieved from https://arxiv.org/abs/1908.06148
[4] Saaim, M., Al-Khateeb, H., & Epiphaniou, G. (2023). File fragment type classification using light-weight convolutional neural networks. *ResearchGate Preprint*, 1–15.
[5] Liu, W., Wang, Y., & Zhang, X. (2024). ByteNet: Rethinking multimedia file fragment classification through visual perspectives. *IEEE Transactions on Multimedia*, 26, 4120–4132.
[6] Joulin, A., Devlin, J., Mikolov, T., & Grave, E. (2020). Comparative evaluation of FastText and BERT for text classification. *Proceedings of the Twelfth Language Resources and Evaluation Conference (LREC 2020)*, 2410–2417.

Kozachenko D. O., Nesterov Y. M.

Analysis of problems and existing solutions for automating file search in game applications

Abstract. This paper analyzes the technical challenges of automated discovery and extraction of text resources, graphical assets, and configurations within compiled containers of modern video games for localization and modding. Existing standard unpackers, byte frequency analysis methods, and AI-based tools for file fragment classification are evaluated, and a combined AI-based system architecture is proposed.

Keywords: *game localization; modding; game directory analysis; file type identification; deep neural networks.*