ДОДАТОК А

Апробація результатів роботи (теза)

АКТУАЛЬНІСТЬ ВІРТУАЛІЗАЦІЇ ГНУЧКИХ ВИРОБНИЧИХ ДІЛЬНИЦЬ НА ВИРОБНИЦТВІ

Берест Б.Р. Гурін Д.В.
Харківський національний університет радіоелектроніки,
Україна, 61166, Харків, пр. Науки 14
E-mail: bohdan.berest@nure.ua, dmytro.gurin@nure.ua.

Анотація: представлено аналіз та концепцію віртуального макета для дослідження гнучкої виробничої дільниці. Проведено огляд сучасних методів автоматизації та цифрового моделювання виробничих процесів, визначено їхні переваги та недоліки. Запропоновано підхід до усунення виявлених недоліків та розглянуто способи його реалізації.

Ключові слова: гнучка виробнича дільниця, автоматизація, управління.

THE RELEVANCE OF VIRTUALIZATION OF A FLEXIBLE MANUFACTURING SYSTEM IN PRODUCTION

Berest B.R. Gurin D.V.
Kharkiv National University of Radio Electronics
Ukraine, 61166, Kharkiv, Nauky av., 14
E-mail: bohdan.berest@nure.ua, dmytro.gurin@nure.ua.

Abstract: The analysis and concept of a virtual model for researching a flexible manufacturing system are presented. A review of modern methods of automation and digital modeling of production processes is conducted, their advantages and disadvantages are identified. An approach to eliminating the identified shortcomings is proposed and methods for its implementation are considered.

Keywords: flexible manufacturing system, automation, management.

Сучасні виробничі процеси характеризуються високою динамічністю та необхідністю швидкої адаптації до змін ринкових умов. У зв’язку з цим гнучкі виробничі дільниці (ГВД) відіграють ключову роль у забезпеченні ефективного виробництва, оскільки дозволяють оперативно реагувати на зміну попиту, налаштовувати обладнання під нові завдання та оптимізувати використання ресурсів.

Однак, незважаючи на високу ефективність, традиційні методи управління ГВД мають певні обмеження. Вони часто залежать від людського фактора, не завжди забезпечують достатню швидкість прийняття рішень і потребують значних витрат на обслуговування. Тому актуальним напрямком є впровадження інтелектуальних систем управління, що базуються на машинному навчанні, Інтернеті речей (IoT) та цифрових двійниках.

Гнучкі виробничі дільниці передбачають використання роботизованих систем, автоматизованих ліній та інтелектуального програмного забезпечення для оптимізації виробничих процесів. В основі їх роботи лежать такі технології, як:

– Автоматизовані системи управління виробництвом (MES) – забезпечують контроль і координацію виробничих операцій.

– Системи планування ресурсів підприємства (ERP) – допомагають інтегрувати всі процеси виробництва в єдину інформаційну систему.

– Цифрові двійники – використовуються для віртуального моделювання процесів, прогнозування їхньої ефективності та виявлення можливих вузьких місць.

Основними перевагами цих систем є підвищена ефективність, можливість оптимізації ресурсів і зменшення впливу людського фактора. Проте існують і недоліки, серед яких висока вартість впровадження, складність інтеграції з існуючими технологіями та потреба в кваліфікованих фахівцях для обслуговування.

З метою усунення вищеперерахованих недоліків пропонується ряд вдосконалень, що можуть підвищити ефективність управління гнучкими виробничими дільницями:

1. Впровадження систем штучного інтелекту (ШІ)

– Використання нейромережевих алгоритмів для аналізу виробничих даних та прогнозування можливих збоїв та результатів.

– Автоматизація прийняття рішень щодо оптимізації робочого навантаження та розподілу ресурсів.

2. Інтеграція Інтернету речей (IoT)

– Використання датчиків для моніторингу стану обладнання та умов роботи в реальному часі.

– Прогнозне технічне обслуговування, що дозволяє мінімізувати простої.

3. Цифрові двійники виробничих процесів

– Моделювання та симуляція виробничих процесів для підбору оптимальних сценаріїв роботи.

– Виявлення слабких місць у системі ще до їх появи у реальному виробництві.

4. Оптимізація логістики та ресурсів

– Впровадження інтелектуальних алгоритмів для управління потоками матеріалів та продукції.

– Зниження втрат сировини та підвищення енергоефективності виробництва.

5. Автоматизоване навчання персоналу

– Використання віртуальної та доповненої реальності для підготовки операторів виробничого процесу.

– Створення інтерактивних навчальних програм на основі аналізу реальних виробничих ситуацій.

Запропоновані вдосконалення сприятимуть підвищенню ефективності управління виробничими дільницями, а саме:

– Зменшення простоїв обладнання – за рахунок прогнозного технічного обслуговування та автоматизованого контролю стану техніки.

– Оптимізація використання ресурсів – за допомогою інтелектуальних алгоритмів планування.

– Підвищення продуктивності – завдяки автоматизації прийняття рішень та мінімізації людського втручання.

– Гнучкість виробництва – можливість швидкої адаптації до змін ринкової ситуації та вимог замовників.

Моделювання та симуляція виробничих процесів є важливими інструментами для аналізу, оптимізації та управління сучасним виробництвом. Одним із поширених методів є дискретно-стохастичне моделювання (DES), яке дозволяє аналізувати процеси, що складаються з подій, які відбуваються в дискретні моменти часу. Цей підхід широко застосовується для дослідження роботи виробничих ліній, оцінки продуктивності обладнання та оптимізації логістичних процесів. Його перевагами є висока деталізація та можливість аналізу змінних навантажень, проте моделі такого типу потребують значних обчислювальних ресурсів, особливо при моделюванні складних систем.

Іншим важливим методом є агентно-орієнтоване моделювання (ABM), де система представлена у вигляді множини автономних агентів, кожен з яких має власні правила поведінки. Цей підхід використовується для аналізу взаємодії працівників з обладнанням, оптимізації потоків матеріалів у логістичних системах та вивчення автономних мобільних роботів. Головними перевагами є гнучкість у моделюванні складних взаємодій та можливість аналізу адаптивних і самоорганізованих систем, але разом з тим метод потребує значних

обчислювальних ресурсів та ретельного налаштування параметрів агентів для досягнення реалістичних результатів.

Системна динаміка (SD) є ще одним підходом, що використовується для аналізу виробничих процесів. Вона дозволяє досліджувати динаміку системи на макрорівні, представляючи її у вигляді потоків і запасів, що змінюються під впливом зворотних зв'язків. Цей метод застосовується для стратегічного планування, оптимізації управлінських рішень та оцінки впливу зовнішніх факторів на виробництво. Він добре підходить для довгострокового прогнозування, однак його недоліком є менша деталізація у порівнянні з іншими методами, що обмежує можливість використання для складних дискретних процесів.

Одним із найсучасніших підходів є гібридне моделювання, яке поєднує кілька методів для забезпечення комплексного аналізу виробничих систем. Наприклад, дискретно-стохастичне моделювання можна інтегрувати з агентним підходом, що дозволяє отримати детальну картину взаємодії технічних і людських факторів у виробничих процесах. Гібридні методи є найбільш гнучкими та точними, проте їх розробка і впровадження є складними і потребують значних ресурсів для реалізації.

Моделювання виробничих процесів дозволяє підприємствам проводити детальний аналіз та оптимізацію без ризику для реального виробництва. Дискретно-стохастичний підхід є найбільш ефективним для потокових виробничих ліній, агентно-орієнтоване моделювання допомагає аналізувати автономні системи, системна динаміка використовується для стратегічного планування, а гібридні методи поєднують переваги різних підходів, що дозволяє отримати найбільш комплексну картину виробничої системи. Подальші дослідження у цій сфері спрямовані на інтеграцію методів моделювання з технологіями штучного інтелекту та Інтернету речей, що відкриває нові можливості для оптимізації виробничих процесів і підвищення їхньої ефективності.

ВИСНОВКИ. Використання сучасних методів моделювання та симуляції виробничих процесів є ключовим етапом у розвитку гнучких виробничих дільниць. Дискретно-стохастичне моделювання дозволяє детально аналізувати процеси, агентно-орієнтовані підходи дають змогу враховувати взаємодію персоналу та обладнання, системна динаміка забезпечує стратегічне планування, а гібридні методи поєднують переваги різних підходів для отримання комплексного аналізу. Інтелектуальні системи управління є перспективним напрямком у розвитку гнучких виробничих дільниць, оскільки вони значно підвищують адаптивність підприємств до змінних умов, знижують витрати на виробництво та покращують якість продукції. Впровадження таких сучасних технологій, як штучний інтелект, Інтернет речей (IoT) та цифрові двійники, сприятиме створенню більш ефективних та гнучких виробничих систем. Подальші дослідження мають бути спрямовані на інтеграцію цих технологій у різні галузі промисловості, а також на оцінку їхньої економічної доцільності, що дозволить оптимізувати виробничі процеси та забезпечити їхню стійкість до змін зовнішнього середовища.

ЛІТЕРАТУРА

1. Козир С., Малієнко А. МОДЕЛЮВАННЯ ПРОЦЕСІВ ВИДОБУВАННЯ ВУГІЛЛЯ В КОМБАЙНОВІЙ ЛАВІ. Автоматизація виробничих процесів. С. 30.

2. Стеценко К. В. ФУНКЦІОНУВАННЯ ГНУЧКИХ ВИРОБНИЧИХ СИСТЕМ. «AUTOMATION AND DEVELOPMENT OF ELECTRONIC DEVICES» ADED-2020 Part 2. С. 186–187.

3. Український державний університет науки і технологій : Дніпровський металургійний інститут.URL: https://nmetau.edu.ua/file/kompiuterne_modeliuvannia_vyrobnychykh_protsesiv_2019.pdf. (дата доступу: 20.01.25)

4. DSpace :: ELAKPI :: Репозитарій КПІ ім. Ігоря Сікорського. URL: https://ela.kpi.ua/server/api/core/bitstreams/d36e2c27-d4bd-4285-bd78-44029dab43db/content. (дата доступу: 1.02.25)

5. Flexible Production - Produce What You Sell, No Stock, No Stress, More Cash!. Flexible Production. URL: https://www.flexibleproduction.com/. (дата доступу: 2.02.25)

6. Hayes A. Flexible Manufacturing System (FMS): Definition and How It Works. Investopedia. URL: https://www.investopedia.com/terms/f/flexible-manufacturing-system.asp. (дата доступу: 10.02.25)

7. Yevsieiev, V., & Gurin, D. (2024). Study of Methods of Dynamic Description of The Environment for Collaborative Robots-Manipulators in the Concepts of Industry 5.0 (Doctoral dissertation, Collection of scientific papers «SCIENTIA»).

8. Abu-Jassar, A. T., Attar, H., Yevsieiev, V., Amer, A., Demska, N., Luhach, A. K., & Lyashenko, V. (2022). Electronic user authentication key for access to HMI/SCADA via unsecured internet networks. Computational intelligence and neuroscience, 2022(1), 5866922.

9. Gurin, D., & et al. (2024). Using Convolutional Neural Networks to Analyze and Detect Key Points of Objects in Image. Multidisciplinary Journal of Science and Technology, 4(9), 5-15.

10. Yevsieiev, V., & et al. (2025). Development of a program for processing 3d models of objects in a collaborative robot workspace using an HD camera. ACUMEN: International journal of multidisciplinary research, 2(1), 194-210.

11. Gurin, D., & et al. (2024). Effect of Frame Processing Frequency on Object Identification Using MobileNetV2 Neural Network for a Mobile Robot. Multidisciplinary Journal of Science and Technology, 4(8), 36-44.

12. Attar, H., Abu-Jassar, A. T., Amer, A., Lyashenko, V., Yevsieiev, V., & Khosravi, M. R. (2022). Control system development and implementation of a CNC laser engraver for environmental use with remote imaging. Computational intelligence and neuroscience, 2022(1), 9140156.

13. Nevliudov, I., & et al.. (2020). Method of Algorithms for Cyber-Physical Production Systems Functioning Synthesis. International Journal of Emerging Trends in Engineering Research, 8(10), 7465-7473.

14. Abu-Jassar AT, Attar H, Amer A, et al. Remote Monitoring System of Patient Status in Social IoT Environments Using Amazon Web Services (AWS) Technologies and Smart Health Care. International Journal of Crowd Science, 2024

15. Abu-Jassar A. Building a Route for a Mobile Robot Based on the BRRT and A*(H-BRRT) Algorithms for the Effective Development of Technological Innovations / Amer Abu-Jassar, Hassan Al-Sukhni, Yasser Al-Sharo, S. Maksymova, V. Yevsieiev, V. Lyashenko // International Journal of Engineering Trends and Technology. – 2024. – V. 72(11). – P. 294-306.

16. Maksymova, S., Yevsieiev, V., & Abu-Jassar, A. (2025). A PROTOTYPE DEVELOPMENT FOR AN AUTOMATED CONTROL SYSTEM FOR PRODUCTION CHECKPOINTS. Multidisciplinary Journal of Science and Technology, 5(3), 287-297.

17. Nimets, K., Abu-Jassar, A., Maksymova, S., & Yevsieiev, V. (2025). A SMALL-SIZED MOBILE ROBOT DEVELOPMENT. ACUMEN: International journal of multidisciplinary research, 2(3), 12-25.

18. Demska, N., Yevsieiev, V., Maksymova, S., & Alkhalaileh, A. (2025). ANALYSIS OF METHODS, MODELS AND ALGORITHMS FOR A COLLABORATIVE ROBOTS GROUP DECENTRALIZED CONTROL. ACUMEN: International journal of multidisciplinary research, 2(2), 235-249.

19. Yevsieiev, V., Maksymova, S., Alkhalaileh, A., & Gurin, D. (2025). Development of a program for processing 3d models of objects in a collaborative robot workspace using an HD camera. ACUMEN: International journal of multidisciplinary research, 2(1), 194-210.

ДОДАТОК Б

Апробація результатів роботи (стаття)

УДК 687.02

ДОСЛІДЖЕННЯ ВИКОРИСТАННЯ ГНУЧКИХ ВИРОБНИЧИХ СИСТЕМ ТА ЇХ КЛАСИФІКАЦІЯ

Б. Р. Берест
Харківський національний університет радіоелектроніки
Україна, 61166, Харків, пр. Науки 14
E-mail: bohdan.berest@nure.ua

Анотація: У даній статті проведено дослідження особливостей використання гнучких виробничих систем (ГВС) у сучасній промисловості. Проаналізовано основні сфери їх застосування, визначено переваги впровадження ГВС у контексті підвищення ефективності виробництва, гнучкості та адаптивності до змін ринкових умов. Також представлено класифікацію гнучких виробничих систем за різними критеріями: ступенем автоматизації, типом продукції, характером управління.

Ключові слова: гнучка виробнича система, автоматизація, класифікація, адаптивність, ефективність.

RESEARCH INTO THE USE OF FLEXIBLE MANUFACTURING SYSTEMS AND THEIR CLASSIFICATION

B. Berest
Kharkiv National University of Radio Electronics
Ukraine, 61166, Kharkiv, 14 Nauky Ave
E-mail: bohdan.berest@nure.ua

Annotation: This article presents a study on the application of Flexible Manufacturing Systems (FMS) in modern industry. The main areas of their implementation are analyzed, along with the key advantages of adopting FMS, such as increased production efficiency, adaptability, and responsiveness to changing market demands. A classification of flexible manufacturing systems is provided based on various criteria, including the level of automation, type of production, and control structure.

Keywords: flexible manufacturing system, automation, classification, adaptability, efficiency.

У сучасних умовах глобалізації та високої конкуренції виробничим підприємствам доводиться шукати нові шляхи підвищення гнучкості та ефективності виробництва. Постійне оновлення продукції, коливання обсягів попиту, персоналізація товарів вимагають від виробничих систем здатності до швидкої адаптації. Одним із найперспективніших рішень у цьому напрямі є впровадження гнучких виробничих систем (ГВС), які дозволяють поєднати автоматизацію з можливістю швидкого переналаштування обладнання.

Гнучка виробнича система (ГВС) – це інтегрований комплекс взаємопов'язаних машин, механізмів, засобів керування та інформаційних технологій, що забезпечує автоматизоване виробництво з можливістю швидкого переналагодження під нові завдання. Основна особливість ГВС полягає в її здатності до адаптації без необхідності повного перепланування або припинення виробничого процесу [2]. Це дозволяє оперативно реагувати на зміни в обсягах замовлень, типах продукції чи навіть у конструкції виробів.

ГВС ґрунтуються на таких ключових принципах:

– автоматизація технологічних та логістичних операцій;

– цифрове управління усіма етапами виробництва;

– модульність і масштабованість, що дозволяє розширювати або змінювати конфігурацію системи;

– інтелектуальне прийняття рішень на базі аналізу даних у реальному часі.

Типова структура ГВС включає:

– Верстати з числовим програмним керуванням (ЧПК):

Це високоточне обладнання, здатне обробляти різноманітні матеріали за заздалегідь заданими програмами. Вони є основними робочими вузлами, що виконують обробку деталей (точіння, фрезерування, свердління тощо). Їхня програмованість дозволяє швидко змінювати тип обробки без фізичного втручання.

– Роботизовані маніпулятори:

Використовуються для переміщення заготовок і готових деталей між різними робочими зонами. Вони можуть виконувати функції завантаження/розвантаження, складання, пакування тощо. Завдяки шести (або більше) ступеням свободи, роботизовані руки забезпечують високу гнучкість та точність маніпуляцій.

– Автоматизовані складальні та транспортні системи:

До них належать конвеєри, рельсові транспортні платформи, автоматизовані складські ліфти, лінії подачі та сортування. Ці системи забезпечують безперервність і синхронізацію виробничого потоку, мінімізуючи простої та втрати часу.

– Єдина система управління та моніторингу:

Центральним елементом є контролер (наприклад, ПЛК – програмований логічний контролер або промисловий комп'ютер), який координує всі підсистеми. Через цю платформу оператори мають змогу змінювати виробничі програми, стежити за станом обладнання та реагувати на позаштатні ситуації. Системи можуть бути локальними або хмарними, з можливістю дистанційного доступу.

– Інтерфейси збору й обробки даних:

Для забезпечення зворотного зв'язку використовуються датчики, сенсори, RFID-мітки, камери машинного зору, IoT-модулі. Вони дозволяють відстежувати температуру, вібрацію, знос інструменту, положення деталей у просторі тощо. Це забезпечує реальну самодіагностику системи та дає змогу проводити превентивне обслуговування [2-3].

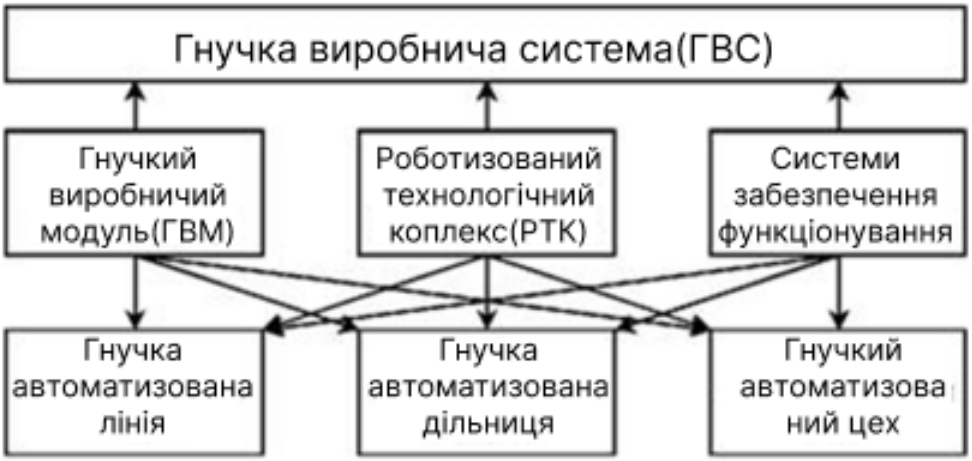

Рисунок 1 – Структура гнучкої виробничої системи

Класифікація гнучких виробничих систем (ГВС) є важливим елементом їх дослідження, оскільки дозволяє систематизувати різновиди таких систем залежно від рівня автоматизації, типу виготовлюваної продукції, а також структури управління [2]. Такий підхід дає змогу краще зрозуміти функціональні можливості ГВС, оцінити їх ефективність у різних умовах виробництва та підібрати оптимальні рішення для впровадження на підприємстві.

Одним із основних критеріїв класифікації є ступінь автоматизації. За цим параметром ГВС поділяються на напівавтоматизовані та повністю автоматизовані. Напівавтоматизовані системи передбачають часткове залучення людини в процес виробництва. Деякі технологічні або допоміжні операції виконуються автоматизованими пристроями, тоді як інші – залишаються під контролем оператора. Такий підхід часто використовується на перехідному етапі до повної автоматизації, а також у випадках, коли повна автоматизація є економічно недоцільною. Наприклад, у дрібносерійному виробництві, де зміна продукції відбувається досить часто, залишення певної гнучкості у вигляді людського втручання може бути більш ефективним.

Натомість повністю автоматизовані ГВС орієнтовані на безперервне виконання виробничих циклів без необхідності участі людини. У таких системах усі елементи – від обробки деталей до їх транспортування і складання – працюють у режимі повної автоматизації. Управління відбувається за допомогою цифрових систем, з використанням ПЛК, промислових комп'ютерів, сенсорних мереж та штучного інтелекту. Подібні рішення особливо ефективні для серійного або масового виробництва, де висока продуктивність та стабільність процесів мають вирішальне значення [4].

Другим важливим критерієм класифікації є тип продукції, що виготовляється в межах системи. Одиничне виробництво передбачає створення індивідуальних виробів, часто з високим рівнем кастомізації або технічної складності. Такі системи застосовуються у галузях, де важлива гнучкість і здатність швидко адаптуватися до нових технічних вимог – наприклад, у авіабудуванні або виробництві дослідних зразків. У цьому випадку ГВС дозволяє забезпечити високу точність і контроль на кожному етапі виготовлення, а можливість перепрограмування обладнання забезпечує оперативність у переході від одного виробу до іншого [2].

Дрібносерійне виробництво об'єднує риси одиничного та серійного – виготовляються невеликі партії однакової продукції, але з частою зміною замовлень. ГВС у такому контексті дає змогу легко здійснювати переналагодження виробничих процесів, використовуючи єдину платформу для обробки широкого спектра виробів.

У серійному виробництві ГВС дозволяє забезпечити масовий випуск стандартних виробів із періодичним оновленням моделей чи конфігурацій. Системи такого типу мають перевагу у стабільності виробництва,

водночас залишаючи можливість для гнучкого налаштування процесів завдяки використанню програмного забезпечення, змінних модулів та автоматичних ліній.

Ще одним важливим критерієм класифікації є тип управління, який використовується в межах гнучкої виробничої системи. Існують три основні типи: централізовані, децентралізовані та гібридні.

Централізовані системи передбачають наявність єдиного центру управління, через який координуються всі виробничі процеси. Такий підхід забезпечує високий рівень контролю, простоту в управлінні та узгоджену взаємодію між усіма компонентами. Проте він має і свої недоліки – зокрема, залежність від єдиного керівного вузла. У разі його виходу з ладу вся система може втратити працездатність або зупинитися. Окрім того, централізовані рішення часто вимагають складного програмного забезпечення для інтеграції великої кількості функцій в одну систему.

Децентралізовані системи побудовані на принципі автономного функціонування окремих модулів. Кожен модуль – обробний, транспортний, або керуючий – має власний контролер, який забезпечує виконання завдань на локальному рівні. Така структура дозволяє зменшити навантаження на центральну систему, підвищує відмовостійкість і полегшує масштабування виробництва. Недоліком може бути складність у синхронізації роботи між модулями, особливо при великій кількості одночасно діючих елементів.

Гібридні системи поєднують елементи централізованого управління з локальними автономними блоками. Це дозволяє досягти балансу між контролем, ефективністю та гнучкістю. Наприклад, центральна система може відповідати за загальне планування і моніторинг, тоді як локальні контролери забезпечують автономне виконання специфічних завдань на окремих етапах. Подібна структура забезпечує високу адаптивність до змін у виробничих умовах, але водночас вимагає ретельного проєктування й налагодження каналів зв'язку, а також ефективного управління інформаційними потоками [4].

У результаті, поділ гнучких виробничих систем за ступенем автоматизації, типом продукції та типом управління дає змогу не лише краще зрозуміти особливості їх функціонування, а й вибрати найбільш ефективну конфігурацію для конкретних умов застосування. Врахування цих класифікаційних ознак є ключовим етапом у процесі проєктування, впровадження та подальшої оптимізації гнучких виробничих систем на сучасних підприємствах.

Гнучкі виробничі системи (ГВС) є ключовим компонентом сучасного промислового середовища, особливо в умовах стрімкого розвитку технологій та зростання вимог до індивідуалізації продукції. Однією з основних переваг таких систем є їхня висока гнучкість та адаптивність. Це означає, що підприємства, які впроваджують ГВС, здатні швидко перебудовувати виробничі процеси залежно від змін у типі продукції або обсягах замовлень, не втрачаючи при цьому ефективності. ГВС дає змогу легко переключатися між різними видами виробів без потреби у тривалій переналадці обладнання, що особливо важливо для підприємств, що працюють у режимі дрібносерійного або індивідуального виробництва.

Окрім гнучкості, ще однією важливою перевагою є оптимізація виробничих витрат. Завдяки високому рівню автоматизації зменшується потреба в людських ресурсах, знижується ймовірність помилок, пов'язаних з людським фактором, і підвищується загальна стабільність процесу [5]. Зменшення витрат стосується також ресурсного споживання, часу циклу виготовлення продукції та вартості технічного обслуговування. Автоматизоване управління процесами дозволяє знизити обсяг відходів, підвищити ефективність використання матеріалів та енергії, а також краще контролювати якість на кожному етапі виробництва.

Ще однією вагомою перевагою є забезпечення високої точності обробки та стабільності якості продукції. За допомогою числового програмного керування, роботизованих систем і вбудованих сенсорів досягається точне дотримання технологічних параметрів, що особливо актуально для галузей із підвищеними вимогами до якості – таких як медицина, електроніка чи авіабудування. Автоматизовані системи дозволяють зменшити варіації між партіями продукції та забезпечити суворе дотримання технічних умов.

Час на переналадку та запуск нових виробничих програм у ГВС значно скорочується порівняно з традиційними системами. Завдяки програмованості устаткування, оператор може швидко змінити параметри виробничого процесу або завантажити нову конфігурацію з цифрового інтерфейсу без потреби у фізичному втручанні. Це забезпечує не тільки економію часу, а й підвищену оперативність підприємства в реагуванні на запити ринку.

Варто також відзначити інтеграцію ГВС з цифровими технологіями, такими як цифрові двійники, інтернет речей (IoT), штучний інтелект, машинне навчання та хмарні сервіси. Це дозволяє створювати розумні виробничі середовища, у яких процеси самостійно аналізуються, оптимізуються й адаптуються у реальному часі [5]. Наприклад, цифрові двійники дають змогу змоделювати виробничу лінію у віртуальному середовищі, протестувати різні сценарії та налаштування до їх впровадження у реальній системі. Інтернет речей забезпечує постійний моніторинг параметрів обладнання, зчитування даних про навантаження, знос або стан навколишнього середовища, що дозволяє здійснювати своєчасне технічне обслуговування та запобігати аваріям.

Проте, попри численні переваги, гнучкі виробничі системи мають і низку недоліків та викликів, які необхідно враховувати при їх впровадженні [2-3]. Однією з основних проблем є висока вартість початкових інвестицій. Встановлення автоматизованого обладнання, програмного забезпечення, мережевої інфраструктури та систем безпеки потребує значних фінансових витрат. Це може стати бар'єром для малих і середніх підприємств, які не мають достатніх фінансових резервів.

Проєктування та впровадження ГВС є складним інженерним завданням. Необхідність врахування широкого спектру технічних аспектів – від вибору обладнання до розробки алгоритмів керування – потребує

участі висококваліфікованих спеціалістів. Такі системи мають бути не лише функціональними, а й стабільними, безпечними та масштабованими, що додає складності на етапі розробки.

Іншим викликом є потреба в добре розвиненій IT-інфраструктурі. ГВС генерує значні обсяги даних, які необхідно зберігати, обробляти й аналізувати [4]. Це вимагає використання серверів, мережевих рішень, спеціалізованого програмного забезпечення, а також заходів із кібербезпеки для захисту даних та управління доступом.

Крім того, у разі використання застарілого або несумісного обладнання можуть виникати труднощі з масштабованістю системи. Оновлення одного модуля не завжди узгоджується з рештою інфраструктури, особливо якщо використовується обладнання від різних виробників або програмне забезпечення з несумісними протоколами. Це створює додаткові труднощі у підтримці системи в актуальному стані.

Таким чином, гнучкі виробничі системи є потужним інструментом для модернізації виробництва, підвищення його ефективності, якості та адаптивності до змін. Проте успішне впровадження ГВС потребує серйозного підходу до планування, наявності технічних і кадрових ресурсів, а також стратегічного бачення щодо розвитку підприємства у цифрову епоху.

ВИСНОВКИ. Гнучкі виробничі системи (ГВС) є важливим компонентом сучасного автоматизованого виробництва, який забезпечує високу адаптивність, ефективність та універсальність виробничих процесів. ГВС дозволяють швидко реагувати на зміни в попиті, скорочувати час на переналадку обладнання, оптимізувати витрати та покращувати якість продукції.

Розглянута класифікація гнучких виробничих систем – за ступенем автоматизації, типом продукції та типом управління – дозволяє глибше зрозуміти їхню внутрішню організацію, функціональні особливості та сфери застосування. Визначені переваги використання таких систем підкреслюють їхню доцільність у контексті Індустрії 4.0 та цифрової трансформації промисловості.

Попри очевидні переваги, впровадження ГВС супроводжується певними викликами, зокрема високою вартістю реалізації, складністю проектування та необхідністю в кваліфікованому персоналі. Проте подальші дослідження й розвиток технологій відкривають нові можливості для оптимізації та масштабування гнучких виробничих систем як у промисловому, так і в освітньо-дослідницькому середовищі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Збірник студентських наукових статей «ADED» | Кафедра комп'ютерно-інтегрованих технологій, автоматизації та робототехніки. *Кафедра комп'ютерно-інтегрованих технологій, автоматизації та робототехніки | Сайт кафедри комп'ютерно-інтегрованих технологій, автоматизації та робототехніки ХНУРЕ*. URL: https://tapr.nure.ua/dijalnist-kafedri/naukova-robota/zbirnik-studentskih-naukovih-statej-avtomatizacija-ta-priladobuduvannja-automation-and-development-of-electronic-devices (дата звернення: 17.04.2025).

2. Особливості створення та експлуатації гнучких виробничих систем. *Pidru4niki*. URL: https://pidru4niki.com/1830050353518/menedzhment/osoblivosti_stvorennya_ekspluatatsiyi_gnuchkih_virobnichih_sistem (дата звернення: 17.04.2025).

3. Нові і специфічні принципи гвс :. *StudFiles*. URL: https://studfile.net/preview/8918303/page:36/ (дата звернення: 17.04.2025).

4. Hayes A. Flexible manufacturing system (FMS): definition and how it works. *Investopedia*. URL: https://www.investopedia.com/terms/f/flexible-manufacturing-system.asp (дата звернення: 17.04.2025).

5. Flexible manufacturing systems: adapt to any conditions. *Autodesk*. URL: https://www.autodesk.com/solutions/flexible-manufacturing-system (date of access: 17.04.2025).

6. *Підручники для вузів онлайн*. URL: https://pidru4niki.com/imag/manag/kap_opm/image038.jpg (дата звернення: 17.04.2025).

7. Chala, O., Yevsieiev, V., Maksymova, S., & Abu-Jassar, A. (2025). MATHEMATICAL MODEL BASED ON MULTI-AGENT REINFORCEMENT LEARNING (MARL) AND PARTIALLY OBSERVABLE MARKOV DECISION PROCESS (POMDP) FOR MODELING CARGO MOVEMENT FOR A MOBILE ROBOTS GROUP. *Multidisciplinary Journal of Science and Technology*, *5*(4), 480-489.

8. Maksymova, S., Yevsieiev, V., & Abu-Jassar, A. (2025). MICROCHIP MARKING RECOGNITION AND IDENTIFICATION USING A COMPUTER VISION SYSTEM MATHEMATICAL MODEL. *Multidisciplinary Journal of Science and Technology*, *5*(4), 321-330.

9. Yevsieiev, V., Maksymova, S., & Alkhalaileh, A. (2025). A METHOD DEVELOPMENT FOR MODELING THE TECHNOLOGICAL PROCESS OF PRINTED CIRCUIT BOARD PRODUCTION BASED ON THE Q-SCHEME. *Multidisciplinary Journal of Science and Technology*, *5*(4), 9-21.

10. Chala, O., Yevsieiev, V., Maksymova, S., & Abu-Jassar, A. (2025). USING THE HUMAN FACE RECOGNITION METHOD BASED ON THE MOBILENETV2 NEURAL NETWORK IN AUTHENTICATION SYSTEMS. *Multidisciplinary Journal of Science and Technology*, *5*(3), 882-895.

11. Maksymova, S., Yevsieiev, V., & Abu-Jassar, A. (2025). A Prototype Development for an Automated Control System for Production Checkpoints. *Multidisciplinary Journal of Science and Technology*, *5*(3), 287-297.

12. Невлюдов, І. Ш., Євсєєв, В. В., & Гурін, Д. В. (2025). MODEL DEVELOPMENT OF DYNAMIC REPRESENTATION A MODEL DESCRIPTION PARAMETERS FOR THE ENVIRONMENT OF A

COLLABORATIVE ROBOT MANIPULATOR WITHIN THE INDUSTRY 5.0 FRAMEWORK. *Системи управління, навігації та зв'язку. Збірник наукових праць*, 1(79), 42-48.

Науковий керівник: *Гурін Дмитро Валерійович, старший викладач кафедри КІТАР. Харківського національного університету радіоелектроніки.*

ДОДАТОК В

Код головної форми «MyForm.h» Visual Studio

```
#pragma comment(lib,"user32.lib")
#pragma comment(lib,"gdi32.lib")
#pragma comment(lib,"glu32.lib")
#pragma comment(lib,"opengl32.lib")

#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <cmath>
#include "MachineQuantityDialog.h"
#include "SettingsForm.h"

int hspos = 50, vspos = 50;
int doorPhase = 0;
int pauseCounter = 0;
int spindlePhase = 0;
int clawsPhase = 0;
int currentCranePosition = 0;
int recordKeyPressCount = 0;
int clawPauseCounter = 0;
int nextMachinePositionIndex = 0;
int numberOfMachinesToDisplay = 0;
int playbackIndex = 0;
int finishedPartsCount = 0;

float zoom = 5.0f;
```

```
float door_scale = 0.0f;
float Rotation = 0;



float spindleAngle = 0.0f;
float spindleRadius = 0.3f;
float dropDistance = 0.3f;
float crane_TargetZ = -2.0f;
float clawsTargetY = 0.0f;
float ropeLength = 0.0f;

bool doorAnimationActive = false;
bool spindleAnimationActive = false;
bool craneMoving = false;
bool clawsMoving = false;
bool objectAttached = false;
bool shouldReleaseObject = false;
bool triedToAttach = false;
bool isRecording = false;
bool isPlaying = false;
bool cranePillarMoving = false;
bool isStanok1Selected = true;

GLfloat cranePillar_TargetX = 0.0f;

GLfloat object_Pos[] = { 0.0f, 1.35f, -7.0f };
GLfloat crane_Translate[] = { 0.0f, 0.0f, -7.0f };
GLfloat crane_pillar_Translate[] = { 0.0f, 0.0f, 0.0f };
GLfloat claws_Translate[] = { 0.0f, 0.0f, -7.0f };
```

```
        GLfloat spindle_Translate[] = { -0.75f, 2.1f, 0.0f };
        GLfloat crossbar_Translate[] = { -0.75f, 2.4f, 0.0f };
        GLfloat pillar_Translate[] = { -0.75f, 2.0f, 0.0f };

        GLfloat cubeColor[] = { 1.0f, 0.0f, 0.0f, 0.5f };

        GLuint floorTextureID;



        const GLfloat OPERATING_X_TARGET = 0.0f;
        const GLfloat INITIAL_SPINDLE_Y = 2.1f;
        const GLfloat INITIAL_CROSSBAR_Y = 2.4f;
        const GLfloat INITIAL_PILLAR_Y = 2.0f;
        const GLfloat REST_X_TARGET = -0.75f;
        const int CHECKERBOARD_SIZE = 256;
        const int TILE_PIXEL_SIZE = 8;

        GLubyte
checkerboardImage[CHECKERBOARD_SIZE][CHECKERBOARD_SIZE][3];
        public ref class MachineInstance {
        public:
            GLfloat x;
            GLfloat y;
            GLfloat z;
            bool isStanok1;
            float rotation;
            float door_scale;
            bool isDoorAnimationActive;
            int doorPhase;
            int pauseCounter;
```

```
array<GLfloat>^ spindle_Translate;
array<GLfloat>^ crossbar_Translate;
array<GLfloat>^ pillar_Translate;
bool spindleAnimationActive;
int spindlePhase;
float spindleAngle;
float spindleRadius;

float dropDistance;
float OPERATING_X_TARGET;
float REST_X_TARGET = -0.75f;
float INITIAL_SPINDLE_Y = 2.1;
float INITIAL_CROSSBAR_Y = 2.4;
float INITIAL_PILLAR_Y = 2.0f;

array<GLfloat>^ cubeColor;

MachineInstance(GLfloat initialX, GLfloat initialY, GLfloat initialZ, bool stanokType)
   : x(initialX), y(initialY), z(initialZ), isStanok1(stanokType),
   isDoorAnimationActive(false), doorPhase(0), door_scale(0.0f), pauseCounter(0),
   spindleAnimationActive(false), spindlePhase(0), spindleAngle(0.0f)
{
   spindleRadius = 0.3f;
   dropDistance = 0.3f;

   spindle_Translate = gcnew array<GLfloat>(3) { REST_X_TARGET, INITIAL_SPINDLE_Y, 0.0f };
```

```
            crossbar_Translate = gcnew array<GLfloat>(3) { REST_X_TARGET, INITIAL_CROSSBAR_Y, 0.0f };
            pillar_Translate = gcnew array<GLfloat>(3) { REST_X_TARGET, INITIAL_PILLAR_Y, 0.0f };

            cubeColor = gcnew array<GLfloat>(4) { 1.0f, 0.0f, 0.0f, 0.5f };
        }
    };

    namespace DD {
        using namespace System;
        using namespace System::ComponentModel;
        using namespace System::Collections;
        using namespace System::Windows::Forms;
        using namespace System::Data;
        using namespace System::Drawing;
        using namespace System::Collections::Generic;
        using namespace std;
        public ref class MyForm : public System::Windows::Forms::Form
        {
        public:
            MyForm(void)
            {
                InitializeComponent();
                InitializeOpenGL();
                renderTimer->Start();
                this->KeyPreview = true;
                InitializeFloorTexture();
                placedMachines = gcnew List<MachineInstance^>();
            }
```

```
protected:
    ~MyForm()
    {
        if (components)
        {
            delete components;
        }
    }
private:
    float pauseLimit = 1000;
    float door_scale_change = 0.01f;
    float move_speed = 0.05f;
    int machine1Stability = 50;
    int machine1Resource = 50;
    int machine2Stability = 50;
    int machine2Resource = 50;
    array<Tuple<GLfloat, GLfloat, GLfloat>^>^ predefinedPositions;
    HDC hdc;
    HGLRC hglrc;
    Timer^ renderTimer;
    Timer^ countTimer;
    Timer^ doorTimer;
    Timer^ spindleTimer;
    Timer^ craneTimer;
    Timer^ clawsTimer;
    Timer^ cranePillarTimer;
    VScrollBar^ vScrollBar1;
    HScrollBar^ hScrollBar1;
    System::ComponentModel::Container^ components;
```

```
        System::Collections::Generic::List<String^>^ recordedActions = gcnew
System::Collections::Generic::List<String^>();
        System::DateTime lastPlaybackTime;
        List<MachineInstance^>^ placedMachines;
        System::Windows::Forms::Button^ btnDeleteLast;
        System::Windows::Forms::Button^ btnDeleteAll;
        System::Windows::Forms::Button^ btnDeleteCount;
        System::Windows::Forms::Button^ btnListCommands;
        System::Windows::Forms::Button^ btnAddStanok;
        System::Windows::Forms::Button^ settingsButton_Click;
    #pragma region Windows Form Designer generated code
        void InitializeComponent(void)
        {
          predefinedPositions = gcnew array<Tuple<GLfloat, GLfloat,
GLfloat>^>{
            Tuple::Create(0.0f, 0.0f, -2.0f),  // Позиція 0
              Tuple::Create(0.0f, 0.0f, 3.0f),   // Позиція 1

              Tuple::Create(4.0f, 0.0f, -2.0f),  // Позиція 2
              Tuple::Create(4.0f, 0.0f, 3.0f),   // Позиція 3
              Tuple::Create(4.0f, 0.0f, -7.0f),  // Позиція 4
              Tuple::Create(4.0f, 0.0f, 7.0f),   // Позиція 5

              Tuple::Create(-4.0f, 0.0f, -2.0f), // Позиція 6
              Tuple::Create(-4.0f, 0.0f, 3.0f),  // Позиція 7
              Tuple::Create(-4.0f, 0.0f, -7.0f), // Позиція 8
              Tuple::Create(-4.0f, 0.0f, 7.0f)   // Позиція 9
          };
          this->vScrollBar1 = (gcnew System::Windows::Forms::VScrollBar());
          this->hScrollBar1 = (gcnew System::Windows::Forms::HScrollBar());
```

```
            this->SuspendLayout();
            this->vScrollBar1->Location = System::Drawing::Point(10, 21);
            this->vScrollBar1->Name = L"vScrollBar1";
            this->vScrollBar1->Size = System::Drawing::Size(20, 570);
            this->vScrollBar1->TabIndex = 0;
            this->vScrollBar1->Value = 50;
            this->vScrollBar1->Scroll += gcnew
System::Windows::Forms::ScrollEventHandler(this, &MyForm::vScrollBar1_Scroll);
            this->hScrollBar1->Location = System::Drawing::Point(47, 10);
            this->hScrollBar1->Name = L"hScrollBar1";
            this->hScrollBar1->Size = System::Drawing::Size(1196, 22);
            this->hScrollBar1->TabIndex = 1;
            this->hScrollBar1->Value = 50;
            this->hScrollBar1->Scroll += gcnew
System::Windows::Forms::ScrollEventHandler(this, &MyForm::hScrollBar1_Scroll);
            this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);
            this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
            this->ClientSize = System::Drawing::Size(1252, 600);
            this->Controls->Add(this->hScrollBar1);
            this->Controls->Add(this->vScrollBar1);
            this->Name = L"MyForm";
            this->Text = L"Simulation";
            this->FormClosing += gcnew
System::Windows::Forms::FormClosingEventHandler(this,
&MyForm::MyForm_Closing);
            this->Load += gcnew System::EventHandler(this,
&MyForm::MyForm_Load);
            this->Resize += gcnew System::EventHandler(this,
&MyForm::MyForm_Resize);
```

```
            this->KeyDown += gcnew KeyEventHandler(this,
&MyForm::MyForm_KeyDown);
            this->MouseWheel += gcnew MouseEventHandler(this,
&MyForm::MyForm_MouseWheel);
            this->btnDeleteLast = (gcnew System::Windows::Forms::Button());
            this->btnDeleteAll = (gcnew System::Windows::Forms::Button());
            this->btnDeleteCount = gcnew System::Windows::Forms::Button();
            this->btnListCommands = gcnew System::Windows::Forms::Button();
            this->btnAddStanok = gcnew System::Windows::Forms::Button();
            this->settingsButton_Click = gcnew
System::Windows::Forms::Button();
            //
            // btnDeleteLast
            //
            this->btnDeleteLast->Location = System::Drawing::Point(40, 90);
            this->btnDeleteLast->Name = L"btnDeleteLast";
            this->btnDeleteLast->Size = System::Drawing::Size(150, 50);
            this->btnDeleteLast->Text = L"Видалити останній верстат";
            this->btnDeleteLast->UseVisualStyleBackColor = true;
            this->btnDeleteLast->Click += gcnew System::EventHandler(this,
&MyForm::btnDeleteLast_Click);
            //
            // btnDeleteAll
            //
            this->btnDeleteAll->Location = System::Drawing::Point(40, 140);
            this->btnDeleteAll->Name = L"btnDeleteAll";
            this->btnDeleteAll->Size = System::Drawing::Size(150, 50);
            this->btnDeleteAll->Text = L"Видалити всі верстати";
            this->btnDeleteAll->UseVisualStyleBackColor = true;
```

```
this->btnDeleteAll->Click += gcnew System::EventHandler(this, &MyForm::btnDeleteAll_Click);


this->btnDeleteCount->Location = System::Drawing::Point(40, 190);
this->btnDeleteCount->Name = L"btnDeleteCount";
this->btnDeleteCount->Size = System::Drawing::Size(150, 50);
this->btnDeleteCount->Text = L"Очистити список готових деталей";
this->btnDeleteCount->UseVisualStyleBackColor = true;
this->btnDeleteCount->Click += gcnew System::EventHandler(this, &MyForm::btnDeleteCount_Click);


this->btnListCommands->Location = System::Drawing::Point(40, 240);
this->btnListCommands->Name = L"btnListCommands";
this->btnListCommands->Size = System::Drawing::Size(150, 50);
this->btnListCommands->Text = L"Список керуючих команд";
this->btnListCommands->UseVisualStyleBackColor = true;
this->btnListCommands->Click += gcnew System::EventHandler(this, &MyForm::btnListCommands_Click);

this->settingsButton_Click->Location = System::Drawing::Point(40, 290);
this->settingsButton_Click->Name = L"settingsButton_Click";
this->settingsButton_Click->Size = System::Drawing::Size(150, 50);
this->settingsButton_Click->Text = L"Налаштування верстатів";
this->settingsButton_Click->UseVisualStyleBackColor = true;
this->settingsButton_Click->Click += gcnew System::EventHandler(this, &MyForm::btnSettingsButton_Click);
```

```
            this->btnAddStanok->Location = System::Drawing::Point(40, 40);
            this->btnAddStanok->Name = L"btnAddStanok";
            this->btnAddStanok->Size = System::Drawing::Size(150, 50);
            this->btnAddStanok->Text = L"Додати верстат";
            this->btnAddStanok->UseVisualStyleBackColor = true;
            this->btnAddStanok->Click += gcnew System::EventHandler(this,
&MyForm::btnAddStanok_Click);

            // Додайте кнопки до форми
            this->Controls->Add(this->btnDeleteLast);
            this->Controls->Add(this->btnDeleteAll);
            this->Controls->Add(this->btnDeleteCount);
            this->Controls->Add(this->btnListCommands);
            this->Controls->Add(this->btnAddStanok);
            this->Controls->Add(this->settingsButton_Click);
            doorTimer = gcnew System::Windows::Forms::Timer();
            doorTimer->Interval = 10;
            doorTimer->Tick += gcnew System::EventHandler(this,
&MyForm::DoorTimer_Tick);
            spindleTimer = gcnew System::Windows::Forms::Timer();
            spindleTimer->Interval = 5;
            spindleTimer->Tick += gcnew System::EventHandler(this,
&MyForm::SpindleTimer_Tick);
            craneTimer = gcnew System::Windows::Forms::Timer();
            craneTimer->Interval = 10;
            craneTimer->Tick += gcnew System::EventHandler(this,
&MyForm::CraneTimer_Tick);
            clawsTimer = gcnew System::Windows::Forms::Timer();
            clawsTimer->Interval = 10;
```

```
            clawsTimer->Tick += gcnew System::EventHandler(this,
&MyForm::ClawsTimer_Tick);
            countTimer = gcnew System::Windows::Forms::Timer();
            countTimer->Interval = 2500;
            countTimer->Tick += gcnew System::EventHandler(this,
&MyForm::playbackTimer_Tick);
            cranePillarTimer = gcnew System::Windows::Forms::Timer();
            cranePillarTimer->Interval = 20;
            cranePillarTimer->Tick += gcnew System::EventHandler(this,
&MyForm::cranePillarTimer_Tick);

            this->ResumeLayout(false);
            this->PerformLayout();
        }
        void InitializeOpenGL() {
            hdc = GetDC((HWND)this->Handle.ToPointer());
            PIXELFORMATDESCRIPTOR pfd;
            ZeroMemory(&pfd, sizeof(pfd));
            pfd.nSize = sizeof(pfd);
            pfd.nVersion = 1;
            pfd.dwFlags = PFD_DRAW_TO_WINDOW |
PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER;
            pfd.iPixelType = PFD_TYPE_RGBA;
            pfd.cColorBits = 32;
            pfd.cDepthBits = 24;
            pfd.iLayerType = PFD_MAIN_PLANE;

            int pixelFormat = ChoosePixelFormat(hdc, &pfd);
            SetPixelFormat(hdc, pixelFormat, &pfd);
            hglrc = wglCreateContext(hdc);
```

```
wglMakeCurrent(hdc, hglrc);
glEnable(GL_DEPTH_TEST);
glClearDepth(1.0f);
glShadeModel(GL_SMOOTH);
glCullFace(GL_BACK);
renderTimer = gcnew Timer();
renderTimer->Interval = 1;
renderTimer->Tick += gcnew EventHandler(this,
&MyForm::RenderScene);
renderTimer->Start();

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

GLfloat globalAmbient[] = { 0.1f, 0.1f, 0.1f, 1.0f };
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, globalAmbient);

GLfloat lightAmbient[] = { 0.2f, 0.2f, 0.2f, 1.0f };
GLfloat lightDiffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };
GLfloat lightSpecular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
GLfloat lightPosition[] = { 5.0f, 5.0f, 5.0f, 1.0f };

glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmbient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDiffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, lightSpecular);
glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);

GLfloat materialSpecular[] = { 0.8f, 0.8f, 0.8f, 1.0f };
```

```
    GLfloat materialShininess[] = { 60.0f };
    glMaterialfv(GL_FRONT, GL_SPECULAR, materialSpecular);
    glMaterialfv(GL_FRONT, GL_SHININESS, materialShininess);

    glEnable(GL_NORMALIZE);

}
void SetupProjection(int width, int height)
{
    if (height == 0) height = 1;
    float aspect = (float)width / (float)height;
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, aspect, 0.1, 50.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
void RenderScene(Object^ sender, EventArgs^ e) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    float cameraX = (float)hspos / 5.0f - 5.0f;
    float cameraY = (float)vspos / 5.0f - 5.0f;
    gluLookAt(cameraX, cameraY, zoom,
        0.0, 0.0, 0.0,
        0.0, 1.0, 0.0);
    for each (MachineInstance ^ machine in placedMachines) {
        glPushMatrix();
        glTranslatef(machine->x, machine->y, machine->z);
```

```
    if (machine->isStanok1) {
        Rotation = machine->rotation;
        door_scale = machine->door_scale;
        DrawBaseStanok();
    }
    else {
        spindle_Translate[0] = machine->spindle_Translate[0];
        spindle_Translate[1] = machine->spindle_Translate[1];
        spindle_Translate[2] = machine->spindle_Translate[2];

        crossbar_Translate[0] = machine->crossbar_Translate[0];
        crossbar_Translate[1] = machine->crossbar_Translate[1];
        crossbar_Translate[2] = machine->crossbar_Translate[2];

        pillar_Translate[0] = machine->pillar_Translate[0];
        pillar_Translate[1] = machine->pillar_Translate[1];
        pillar_Translate[2] = machine->pillar_Translate[2];

        DrawBaseStanok2();
    }

    glPopMatrix();
}
if (objectAttached) {
    object_Pos[0] = claws_Translate[0];
    object_Pos[1] = claws_Translate[1] + 3.6f;
    object_Pos[2] = crane_Translate[2];
}
else {
```

```
            object_Pos[0] = crane_pillar_Translate[0];
        }
        object();
        Crane();
        Crane_claws();
        Crane_cart();
        Elements();
        Cage();
        Floor();
        glFinish();
        SwapBuffers(hdc);
    }

    void makeCheckerboard() {
        for (int i = 0; i < CHECKERBOARD_SIZE; i++) {
            for (int j = 0; j < CHECKERBOARD_SIZE; j++) {
                GLubyte color = (((i / TILE_PIXEL_SIZE) % 2) == ((j / TILE_PIXEL_SIZE) % 2)) ? 255 : 0;

                checkerboardImage[i][j][0] = color;
                checkerboardImage[i][j][1] = color;
                checkerboardImage[i][j][2] = color;
            }
        }
    }
    void InitializeFloorTexture() {
        makeCheckerboard();
        glGenTextures(1, &floorTextureID);
        glBindTexture(GL_TEXTURE_2D, floorTextureID);
```

```
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
            glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB,
CHECKERBOARD_SIZE, CHECKERBOARD_SIZE, 0, GL_RGB,
GL_UNSIGNED_BYTE, checkerboardImage);
            gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB,
CHECKERBOARD_SIZE, CHECKERBOARD_SIZE, GL_RGB,
GL_UNSIGNED_BYTE, checkerboardImage);
        }

        MachineInstance^ FindMachineWithObject() {
            float horizontalProximityThreshold = 1.5f;
            float verticalProximityThreshold = 2.0f;
            for each (MachineInstance ^ machine in placedMachines) {
                float dx = fabs(machine->x - object_Pos[0]);
                float dz = fabs(machine->z - object_Pos[2]);
                float dy = fabs(machine->y - object_Pos[1]);
                if (dx < horizontalProximityThreshold && dz <
horizontalProximityThreshold) {
                    if (dy < verticalProximityThreshold) {
                        return machine;
                    }
                }
            }
```

```
    return nullptr;
}
MachineInstance^ FindMachineWithSpindleAnimation() {
    for each (MachineInstance ^ machine in placedMachines) {
        if (!machine->isStanok1 && machine->spindleAnimationActive) {
            return machine;
        }
    }
    return nullptr;
}

void Cube(float size) {
    float halfSize = size / 2.0f;

    glBegin(GL_QUADS);

    // Верхня грань (Y+)
    glNormal3f(0.0f, 1.0f, 0.0f); // Нормаль для верхньої грані
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-halfSize, halfSize, -halfSize);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(halfSize, halfSize, -halfSize);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(halfSize, halfSize, halfSize);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-halfSize, halfSize, halfSize);

    // Нижня грань (Y-)
    glNormal3f(0.0f, -1.0f, 0.0f); // Нормаль для нижньої грані
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-halfSize, -halfSize, -halfSize);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(halfSize, -halfSize, -halfSize);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(halfSize, -halfSize, halfSize);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-halfSize, -halfSize, halfSize);
```

```
    // Передня грань (Z+)
    glNormal3f(0.0f, 0.0f, 1.0f); // Нормаль для передньої грані
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-halfSize, -halfSize, halfSize);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(halfSize, -halfSize, halfSize);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(halfSize, halfSize, halfSize);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-halfSize, halfSize, halfSize);

    // Задня грань (Z-)
    glNormal3f(0.0f, 0.0f, -1.0f); // Нормаль для задньої грані
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-halfSize, -halfSize, -halfSize);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(halfSize, -halfSize, -halfSize);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(halfSize, halfSize, -halfSize);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-halfSize, halfSize, -halfSize);

    // Права грань (X+)
    glNormal3f(1.0f, 0.0f, 0.0f); // Нормаль для правої грані
    glTexCoord2f(0.0f, 0.0f); glVertex3f(halfSize, -halfSize, -halfSize);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(halfSize, halfSize, -halfSize);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(halfSize, halfSize, halfSize);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(halfSize, -halfSize, halfSize);

    // Ліва грань (X-)
    glNormal3f(-1.0f, 0.0f, 0.0f); // Нормаль для лівої грані
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-halfSize, -halfSize, -halfSize);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-halfSize, halfSize, -halfSize);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-halfSize, halfSize, halfSize);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-halfSize, -halfSize, halfSize);

    glEnd();
}
```

```
void Celinder(float baseRadius, float topRadius, float height, int slices, int
stacks) {
    GLUquadric* quadric = gluNewQuadric();
    gluQuadricTexture(quadric, GL_TRUE);
    glPushMatrix();
    glTranslatef(0.0f, 0.0f, 0.0f);
    gluDisk(quadric, 0.0f, baseRadius, slices, 1);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(0.0f, 0.0f, height);
    gluDisk(quadric, 0.0f, topRadius, slices, 1);
    glPopMatrix();
    gluCylinder(quadric, baseRadius, topRadius, height, slices, stacks);
    gluDeleteQuadric(quadric);
}

void Floor() {
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, floorTextureID);

    glPushMatrix();
    glTranslatef(0.0f, -0.1f, 0.0f);
    glScalef(15.0f, 0.1f, 17.5f);
    glColor3f(1.0f, 1.0f, 1.0f);
    Cube(1.0f);
    glPopMatrix();

    glDisable(GL_TEXTURE_2D);
}
void object() {
```

```
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    glPushMatrix();
    glTranslatef(object_Pos[0], object_Pos[1], object_Pos[2]);
    glScalef(0.3f, 0.3f, 0.3f);
    glColor4f(cubeColor[0], cubeColor[1], cubeColor[2], cubeColor[3]);
    Cube(1.0f);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    Cube(1.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glPopMatrix();

    glDisable(GL_BLEND);
}
void DrawBaseStanok2() {
    glPushMatrix();
    glTranslatef(0.0f, 0.7f, 0.0f);
    glScalef(2.0f, 1.0f, 3.0f);
    glColor3f(0.5f, 0.5f, 0.5f);
    Cube(1.0f);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    Cube(1.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(0.0f, 0.0f, 1.3f);
```

```
glScalef(2.0f, 0.4f, 0.4f);
glColor3f(1.0f, 0.7f, 0.0f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(0.0f, 0.0f, -1.3f);
glScalef(2.0f, 0.4f, 0.4f);
glColor3f(1.0f, 0.7f, 0.0f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(0.0f, 1.35f, 1.3f);
glScalef(2.0f, 0.3f, 0.4f);
glColor3f(1.0f, 0.3f, 0.0f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();
```

```
glPushMatrix();
glTranslatef(0.0f, 1.35f, -1.3f);
glScalef(2.0f, 0.3f, 0.4f);
glColor3f(1.0f, 0.3f, 0.0f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(pillar_Translate[0], pillar_Translate[1], -1.3f);
glScalef(0.2f, 1.0f, 0.2f);
glColor3f(1.0f, 0.3f, 0.0f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(pillar_Translate[0], pillar_Translate[1], 1.3f);
glScalef(0.2f, 1.0f, 0.2f);
glColor3f(1.0f, 0.3f, 0.0f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
```

```
            Cube(1.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
            glPopMatrix();

            glPushMatrix();
            glTranslatef(crossbar_Translate[0], crossbar_Translate[1],
crossbar_Translate[2]);
            glScalef(0.2f, 0.2f, 3.4f);
            glColor3f(1.0f, 0.3f, 0.0f);
            Cube(1.0f);
            glColor3f(0.0f, 0.0f, 0.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
            Cube(1.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
            glPopMatrix();

            glPushMatrix();
            glTranslatef(spindle_Translate[0], spindle_Translate[1] + 0.1f,
spindle_Translate[2]);
            glScalef(0.2f, 0.2f, 0.2f);
            glColor3f(1.0f, 1.0f, 0.0f);
            Cube(1.0f);
            glColor3f(0.0f, 0.0f, 0.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
            Cube(1.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
            glPopMatrix();

            glPushMatrix();
```

```
            glTranslatef(spindle_Translate[0], spindle_Translate[1],
spindle_Translate[2]);
            glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
            glColor3f(0.5f, 0.5f, 0.5f);
            Celinder(0.05f, 0.05f, 0.1f, 32, 32);
            glColor3f(0.0f, 0.0f, 0.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
            Celinder(0.05f, 0.05f, 0.1f, 8, 8);
            glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
            glPopMatrix();
        }
        void DrawBaseStanok() {
            glEnable(GL_BLEND);
            glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

            glPushMatrix();
            glTranslatef(0.0f, 0.7f, 0.0f);
            glScalef(2.0f, 1.0f, 3.0f);
            glColor3f(0.5f, 0.5f, 0.5f);
            Cube(1.0f);
            glColor3f(0.0f, 0.0f, 0.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
            Cube(1.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
            glPopMatrix();

            glPushMatrix();
            glTranslatef(0.0f, 0.0f, 1.3f);
            glScalef(2.0f, 0.4f, 0.4f);
            glColor3f(1.0f, 0.7f, 0.0f);
```

```
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(0.0f, 0.0f, -1.3f);
glScalef(2.0f, 0.4f, 0.4f);
glColor3f(1.0f, 0.7f, 0.0f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(0.0f, 1.7f, -1.15f);
glScalef(2.0f, 1.0f, 0.7f);
glColor3f(0.8f, 0.8f, 0.8f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
```

```
glTranslatef(0.0f, 1.7f, 1.15f);
glScalef(2.0f, 1.0f, 0.7f);
glColor3f(0.8f, 0.8f, 0.8f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(0.0f, 2.3f, 1.0f - (2 * door_scale));
glScalef(2.0f, 0.1f, 0.8f);
glColor4f(1.0f, 0.0f, 0.0f, 0.2f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(0.0f, 2.3f, -1.0f + (2 * door_scale));
glScalef(2.0f, 0.1f, 0.8f);
glColor4f(1.0f, 0.0f, 0.0f, 0.2f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
```

```
glPopMatrix();

glPushMatrix();
glTranslatef(-0.9f, 1.7f, 0.0f);
glScalef(0.2f, 1.0f, 1.58f);
glColor3f(0.8f, 0.8f, 0.8f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(1.0f, 1.7f, 1.0f - (2 * door_scale));
glScalef(0.1f, 1.1f, 0.8f);
glColor4f(1.0f, 0.0f, 0.0f, 0.2f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(1.0f, 1.7f, -1.0f + (2 * door_scale));
glScalef(0.1f, 1.1f, 0.8f);
glColor4f(1.0f, 0.0f, 0.0f, 0.2f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
```

```
            glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
            Cube(1.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
            glPopMatrix();

            glPushMatrix();
            glTranslatef(0.2f, 1.7f, -0.8f);
            glRotatef(Rotation, 0.0f, 0.0f, 1.0f);
            glColor3f(1.0f, 0.3f, 0.5f);
            Celinder(0.2f, 0.2f, 0.2f, 32, 32);
            glColor3f(0.0f, 0.0f, 0.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
            Celinder(0.2f, 0.2f, 0.2f, 8, 8);
            glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
            glPopMatrix();

            glDisable(GL_BLEND);
        }
        void Crane_claws() {
            glPushMatrix();

            glTranslatef(crane_Translate[0], crane_Translate[1],
crane_Translate[2]);

            glPushMatrix();
            glTranslatef(0.0f, 4.75f, 0.0f);
            glScalef(0.65f, 0.65f, 1.0f);
            glColor3f(0.0f, 0.0f, 1.0f);
            Cube(1.0f);
            glColor3f(0.0f, 0.0f, 0.0f);
```

```
            glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
            Cube(1.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
            glPopMatrix();
            glPopMatrix();
        }
        void Crane_cart() {
            glPushMatrix();
            glTranslatef(claws_Translate[0], claws_Translate[1],
claws_Translate[2]);

            glPushMatrix();
            glTranslatef(0.0f, 4.0f, 0.0f);
            glScalef(0.4f, 0.2f, 0.4f);
            glColor3f(1.0f, 0.84f, 0.0f);
            Cube(1.0f);
            glColor3f(0.0f, 0.0f, 0.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
            Cube(1.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
            glPopMatrix();

            glPushMatrix();
            glTranslatef(0.0f, 4.5 + ropeLength, 0.1f);
            glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
            glColor3f(0.68f, 0.85f, 0.90f);
            Celinder(0.035f, 0.035f, 0.4f + ropeLength, 32, 32);
            glColor3f(0.0f, 0.0f, 0.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
            Celinder(0.035f, 0.035f, 0.4f + ropeLength, 6, 6);
```

```
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(0.0f, 4.5 + ropeLength, -0.1f);
glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
glColor3f(0.68f, 0.85f, 0.90f);
Celinder(0.035f, 0.035f, 0.4f + ropeLength, 32, 32);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Celinder(0.035f, 0.035f, 0.4f + ropeLength, 6, 6);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(0.0f, 3.9f, 0.25f);
glScalef(0.035f, 0.3f, 0.035f);
glColor3f(1.0f, 0.3f, 0.0f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(0.0f, 3.9f, -0.25f);
glScalef(0.035f, 0.3f, 0.035f);
glColor3f(1.0f, 0.3f, 0.0f);
Cube(1.0f);
```

```
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(0.25f, 3.9f, 0.0f);
glScalef(0.035f, 0.3f, 0.035f);
glColor3f(1.0f, 0.3f, 0.0f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(-0.25f, 3.9f, 0.0f);
glScalef(0.035f, 0.3f, 0.035f);
glColor3f(1.0f, 0.3f, 0.0f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(-0.23f, 3.75f, 0.0f);
```

```
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glScalef(0.035f, 0.1f, 0.035f);
glColor3f(1.0f, 0.3f, 0.0f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(0.23f, 3.75f, 0.0f);
glRotatef(-45.0f, 0.0f, 0.0f, 1.0f);
glScalef(0.035f, 0.1f, 0.035f);
glColor3f(1.0f, 0.3f, 0.0f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(0.0f, 3.75f, -0.23f);
glRotatef(-45.0f, 1.0f, 0.0f, 0.0f);
glScalef(0.035f, 0.1f, 0.035f);
glColor3f(1.0f, 0.3f, 0.0f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
```

```
    Cube(1.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(0.0f, 3.75f, 0.23f);
    glRotatef(45.0f, 1.0f, 0.0f, 0.0f);
    glScalef(0.035f, 0.1f, 0.035f);
    glColor3f(1.0f, 0.3f, 0.0f);
    Cube(1.0f);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    Cube(1.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glPopMatrix();

    glPopMatrix();
}
void Crane() {
    glPushMatrix();
    glTranslatef(0.0f, 4.75f, -9.3f);
    glScalef(10.0f, 0.5f, 0.5f);
    glColor3f(1.0f, 0.3f, 0.0f);
    Cube(1.0f);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    Cube(1.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glPopMatrix();
```

```
            glPushMatrix();
            glTranslatef(0.0f, 4.75f, 9.3f);
            glScalef(10.0f, 0.5f, 0.5f);
            glColor3f(1.0f, 0.3f, 0.0f);
            Cube(1.0f);
            glColor3f(0.0f, 0.0f, 0.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
            Cube(1.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
            glPopMatrix();

            glPushMatrix();
            glTranslatef(crane_pillar_Translate[0], crane_pillar_Translate[1],
crane_pillar_Translate[2]);

            glPushMatrix();
            glTranslatef(0.0f, 4.75f, 0.0f);
            glScalef(0.5f, 0.5f, 19.1f);
            glColor3f(1.0f, 0.3f, 0.0f);
            Cube(1.0f);
            glColor3f(0.0f, 0.0f, 0.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
            Cube(1.0f);
            glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
            glPopMatrix();

            glPopMatrix();
        }

        void Elements() {
```

```
glPushMatrix();
glRotatef(90, 0.0f, 1.0f, 0.0f);
glTranslatef(7.0f, 0.0f, 0.0f);

glPushMatrix();
glTranslatef(0.0f, 0.7f, 0.0f);
glScalef(2.0f, 1.0f, 3.0f);
glColor3f(0.5f, 0.5f, 0.5f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(0.0f, 0.0f, 1.3f);
glScalef(2.0f, 0.4f, 0.4f);
glColor3f(1.0f, 0.7f, 0.0f);
Cube(1.0f);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Cube(1.0f);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glPopMatrix();

glPushMatrix();
glTranslatef(0.0f, 0.0f, -1.3f);
glScalef(2.0f, 0.4f, 0.4f);
glColor3f(1.0f, 0.7f, 0.0f);
```

```
    Cube(1.0f);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    Cube(1.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glPopMatrix();

    glPopMatrix();
}
void Cage() {
    glPushMatrix();
    glRotatef(90, 0.0f, 1.0f, 0.0f);
    glTranslatef(-7.0f, 0.0f, 0.0f);

    glPushMatrix();
    glTranslatef(0.0f, 0.7f, 0.0f);
    glScalef(2.0f, 1.0f, 3.0f);
    glColor3f(0.5f, 0.5f, 0.5f);
    Cube(1.0f);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    Cube(1.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(0.0f, 0.0f, 1.3f);
    glScalef(2.0f, 0.4f, 0.4f);
    glColor3f(1.0f, 0.7f, 0.0f);
    Cube(1.0f);
```

```
    glColor3f(0.0f, 0.0f, 0.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    Cube(1.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(0.0f, 0.0f, -1.3f);
    glScalef(2.0f, 0.4f, 0.4f);
    glColor3f(1.0f, 0.7f, 0.0f);
    Cube(1.0f);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    Cube(1.0f);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glPopMatrix();

    glPopMatrix();
}

void DoorTimer_Tick(System::Object^ sender, System::EventArgs^ e) {
    MachineInstance^ targetMachine = FindMachineWithObject();
    if (targetMachine == nullptr) {
        doorTimer->Stop();
        return;
    }

    if (targetMachine->doorPhase == 0) {
        targetMachine->door_scale += door_scale_change;
        if (targetMachine->door_scale >= 0.3f) {
```

```
                targetMachine->door_scale = 0.3f;
                targetMachine->doorPhase = 1;
            }
        }
        else if (targetMachine->doorPhase == 1) {
            targetMachine->pauseCounter += doorTimer->Interval;
            if (targetMachine->pauseCounter >= pauseLimit) {
                targetMachine->doorPhase = 2;
            }
        }
        else if (targetMachine->doorPhase == 2) {
            targetMachine->door_scale -= door_scale_change;
            if (targetMachine->door_scale <= 0.0f) {
                targetMachine->door_scale = 0.0f;
                doorTimer->Stop();
                targetMachine->isDoorAnimationActive = false;
            }
            cubeColor[0] = 0.0f;
            cubeColor[1] = 1.0f;
            cubeColor[2] = 0.0f;
            cubeColor[3] = 0.5f;
        }
        this->RenderScene(nullptr, nullptr);
    }

    void SpindleTimer_Tick(System::Object^ sender, System::EventArgs^ e) {
        MachineInstance^ targetMachine = FindMachineWithSpindleAnimation();
        if (targetMachine == nullptr || !targetMachine->spindleAnimationActive) {
```

```
            spindleTimer->Stop();
            return;
        }

        if (targetMachine->spindlePhase == 0) {
            bool arrivedAtOperatingX = true;
            if (targetMachine->spindle_Translate[0] < OPERATING_X_TARGET) {
                targetMachine->spindle_Translate[0] += move_speed;
                if (targetMachine->spindle_Translate[0] > OPERATING_X_TARGET) targetMachine->spindle_Translate[0] = OPERATING_X_TARGET;
                arrivedAtOperatingX = false;
            }
            else if (targetMachine->spindle_Translate[0] > OPERATING_X_TARGET) {
                targetMachine->spindle_Translate[0] -= move_speed;
                if (targetMachine->spindle_Translate[0] < OPERATING_X_TARGET) targetMachine->spindle_Translate[0] = OPERATING_X_TARGET;
                arrivedAtOperatingX = false;
            }
            targetMachine->crossbar_Translate[0] = targetMachine->spindle_Translate[0];
            targetMachine->pillar_Translate[0] = targetMachine->spindle_Translate[0];

            targetMachine->spindle_Translate[2] = 0.0f;
            targetMachine->crossbar_Translate[2] = 0.0f;
            targetMachine->pillar_Translate[2] = 0.0f;
```

```
            if (arrivedAtOperatingX) {
                targetMachine->spindlePhase = 1;
            }
        }
        else if (targetMachine->spindlePhase == 1) {
            bool allLowered = true;
            if (targetMachine->spindle_Translate[1] > (INITIAL_SPINDLE_Y -
dropDistance)) {
                targetMachine->spindle_Translate[1] -= move_speed;
                allLowered = false;
            }
            else {
                targetMachine->spindle_Translate[1] = INITIAL_SPINDLE_Y -
dropDistance;
            }

            if (targetMachine->crossbar_Translate[1] >
(INITIAL_CROSSBAR_Y - dropDistance)) {
                targetMachine->crossbar_Translate[1] -= move_speed;
                allLowered = false;
            }
            else {
                targetMachine->crossbar_Translate[1] = INITIAL_CROSSBAR_Y
- dropDistance;
            }

            if (targetMachine->pillar_Translate[1] > (INITIAL_PILLAR_Y -
dropDistance)) {
                targetMachine->pillar_Translate[1] -= move_speed;
```

```
                allLowered = false;
            }
            else {
                targetMachine->pillar_Translate[1] = INITIAL_PILLAR_Y -
dropDistance;
            }

            if (allLowered) {
                targetMachine->spindlePhase = 2;
            }
        }
        else if (targetMachine->spindlePhase == 2) {
            targetMachine->spindleAngle += 0.05f;
            if (targetMachine->spindleAngle >= 2 * 3.14159f) {
                targetMachine->spindleAngle = 0.0f;
                targetMachine->spindlePhase = 3;
            }
            GLfloat x = OPERATING_X_TARGET + spindleRadius *
cos(targetMachine->spindleAngle);
            GLfloat z = spindleRadius * sin(targetMachine->spindleAngle);
            targetMachine->spindle_Translate[0] = x;
            targetMachine->spindle_Translate[2] = z;
            targetMachine->crossbar_Translate[0] = x;
            targetMachine->crossbar_Translate[2] = z;
            targetMachine->pillar_Translate[0] = x;
            targetMachine->pillar_Translate[2] = z;
        }
        else if (targetMachine->spindlePhase == 3) {
            cubeColor[0] = 0.0f;
            cubeColor[1] = 0.0f;
```

```
                cubeColor[2] = 1.0f;
                cubeColor[3] = 0.5f;

                bool allReset = true;
                if (targetMachine->spindle_Translate[1] < INITIAL_SPINDLE_Y) {
                    targetMachine->spindle_Translate[1] += move_speed;
                    if (targetMachine->spindle_Translate[1] > INITIAL_SPINDLE_Y)
targetMachine->spindle_Translate[1] = INITIAL_SPINDLE_Y;
                    allReset = false;
                }
                if (targetMachine->crossbar_Translate[1] <
INITIAL_CROSSBAR_Y) {
                    targetMachine->crossbar_Translate[1] += move_speed;
                    if (targetMachine->crossbar_Translate[1] >
INITIAL_CROSSBAR_Y) targetMachine->crossbar_Translate[1] =
INITIAL_CROSSBAR_Y;
                    allReset = false;
                }
                if (targetMachine->pillar_Translate[1] < INITIAL_PILLAR_Y) {
                    targetMachine->pillar_Translate[1] += move_speed;
                    if (targetMachine->pillar_Translate[1] > INITIAL_PILLAR_Y)
targetMachine->pillar_Translate[1] = INITIAL_PILLAR_Y;
                    allReset = false;
                }
                if (targetMachine->spindle_Translate[0] > REST_X_TARGET) {
                    targetMachine->spindle_Translate[0] -= move_speed;
                    if (targetMachine->spindle_Translate[0] < REST_X_TARGET)
targetMachine->spindle_Translate[0] = REST_X_TARGET;
                    allReset = false;
                }
```

```
            else if (targetMachine->spindle_Translate[0] < REST_X_TARGET) {
                targetMachine->spindle_Translate[0] += move_speed;
                if (targetMachine->spindle_Translate[0] > REST_X_TARGET)
targetMachine->spindle_Translate[0] = REST_X_TARGET;
                allReset = false;
            }

            targetMachine->crossbar_Translate[0] = targetMachine-
>spindle_Translate[0];
            targetMachine->pillar_Translate[0] = targetMachine-
>spindle_Translate[0];

            targetMachine->spindle_Translate[2] = 0.0f;
            targetMachine->crossbar_Translate[2] = 0.0f;
            targetMachine->pillar_Translate[2] = 0.0f;

            if (allReset &&
                fabs(targetMachine->spindle_Translate[0] - REST_X_TARGET) <
0.01f &&
                fabs(targetMachine->crossbar_Translate[0] - REST_X_TARGET)
< 0.01f &&
                fabs(targetMachine->pillar_Translate[0] - REST_X_TARGET) <
0.01f &&
                fabs(targetMachine->spindle_Translate[1] -
INITIAL_SPINDLE_Y) < 0.01f &&
                fabs(targetMachine->crossbar_Translate[1] -
INITIAL_CROSSBAR_Y) < 0.01f &&
                fabs(targetMachine->pillar_Translate[1] - INITIAL_PILLAR_Y) <
0.01f)
            {
```

```
            spindleTimer->Stop();
            targetMachine->spindleAnimationActive = false;
            targetMachine->spindlePhase = 0;
        }
    }
    this->RenderScene(nullptr, nullptr);
}

void CraneTimer_Tick(System::Object^ sender, System::EventArgs^ e) {
    if (!craneMoving) return;
    float speed = 0.1f;
    float diff = crane_TargetZ - crane_Translate[2];
    if (fabs(diff) < speed) {
        crane_Translate[2] = crane_TargetZ;
        claws_Translate[2] = crane_TargetZ;
        craneMoving = false;
        craneTimer->Stop();
    }
    else {
        crane_Translate[2] += (diff > 0 ? speed : -speed);
        claws_Translate[2] += (diff > 0 ? speed : -speed);
    }

    this->RenderScene(nullptr, nullptr);
}

void ClawsTimer_Tick(System::Object^ sender, System::EventArgs^ e) {
    float clawsSpeed = 0.05f;

    if (clawsPhase == 1) {
```

```
    if (claws_Translate[1] > -2.3f) {
        claws_Translate[1] -= clawsSpeed;
        if (claws_Translate[1] < -2.3f) claws_Translate[1] = -2.3f;
    }
    else {
        clawPauseCounter++;

        if (clawPauseCounter >= 2 && !triedToAttach) {
            float dx = fabs(claws_Translate[0] - object_Pos[0]);
            float dz = fabs(crane_Translate[2] - object_Pos[2]);
            float dy = fabs((claws_Translate[1] + 3.6f) - object_Pos[1]);

            if (dx < 0.3f && dy < 0.3f && dz < 0.3f) {
                objectAttached = true;
            }

            triedToAttach = true;
            clawsPhase = 2;

            if (objectAttached && shouldReleaseObject) {

                objectAttached = false;
                object_Pos[1] = 1.35f;
                object_Pos[0] = claws_Translate[0];
                object_Pos[2] = crane_Translate[2];
                shouldReleaseObject = false;
            }
        }
    }
}
```

```
        else if (clawsPhase == 2) {
            if (claws_Translate[1] < 0.0f) {
                claws_Translate[1] += clawsSpeed;
                if (claws_Translate[1] > 0.0f) claws_Translate[1] = 0.0f;
            }
            else {
                clawsPhase = 0;
                clawsMoving = false;
                triedToAttach = false;
                clawPauseCounter = 0;
                clawsTimer->Stop();
            }
        }
        ropeLength = fabs(claws_Translate[1]);
        this->RenderScene(nullptr, nullptr);
    }
    void cranePillarTimer_Tick(System::Object^ sender, System::EventArgs^
e) {
        if (!cranePillarMoving) return;

        float currentX = crane_pillar_Translate[0];
        float targetX = cranePillar_TargetX;
        float moveSpeed = 0.1f;

        if (currentX < targetX) {
            currentX += moveSpeed;
            if (currentX > targetX) currentX = targetX;
        }
        else if (currentX > targetX) {
            currentX -= moveSpeed;
```

```
            if (currentX < targetX) currentX = targetX;
        }
        crane_pillar_Translate[0] = currentX;
        crane_Translate[0] = currentX;
        claws_Translate[0] = currentX;
        if (fabs(currentX - targetX) < 0.001f) {
            crane_pillar_Translate[0] = targetX;
            crane_Translate[0] = targetX;
            claws_Translate[0] = targetX;
            cranePillarMoving = false;
            cranePillarTimer->Stop();
        }
        this->RenderScene(nullptr, nullptr);
    }

    void recordAction(String^ action) {
        if (isRecording && recordedActions != nullptr) {
            recordedActions->Add(action);
        }
    }
    void playbackTimer_Tick(System::Object^ sender, System::EventArgs^ e)
{
        if (recordedActions->Count == 0) {
            isPlaying = false;
            countTimer->Stop();
            MessageBox::Show("Відтворення закінчено (дії відсутні)",
"Інформація");
            return;
        }
        if (playbackIndex >= recordedActions->Count) {
```

```
            playbackIndex = 0;
        }
        String^ action = recordedActions[playbackIndex];
        if (action->Equals("A")) {
            MachineInstance^ targetStanok1 = FindMachineWithObject();
            if (targetStanok1 != nullptr && targetStanok1->isStanok1 &&
!targetStanok1->isDoorAnimationActive) {
                targetStanok1->isDoorAnimationActive = true;
                targetStanok1->doorPhase = 0;
                targetStanok1->pauseCounter = 0;
                doorTimer->Start();
            }
        }
        else if (action->Equals("B")) {
            MachineInstance^ targetStanok2 = FindMachineWithObject();
            if (targetStanok2 != nullptr && !targetStanok2->isStanok1 &&
!targetStanok2->spindleAnimationActive) {
                targetStanok2->spindleAnimationActive = true;
                targetStanok2->spindlePhase = 0;
                targetStanok2->spindleAngle = 0.0f;
                spindleTimer->Start();
            }
        }
        else if (action->Equals("C")) {
            if (!clawsMoving) {
                clawsMoving = true;
                clawsPhase = 1;
                clawsTimer->Start();
            }
        }
```

```
else if (action->Equals("1")) {
    crane_TargetZ = -7.0f;
    craneMoving = true;
    craneTimer->Start();
}
else if (action->Equals("2")) {
    crane_TargetZ = -2.0f;
    craneMoving = true;
    shouldReleaseObject = true;
    craneTimer->Start();
}
else if (action->Equals("3")) {
    crane_TargetZ = 3.0f;
    craneMoving = true;
    shouldReleaseObject = true;
    craneTimer->Start();
}
else if (action->Equals("4")) {
    crane_TargetZ = 7.0f;
    craneMoving = true;
    shouldReleaseObject = true;
    craneTimer->Start();
}
else if (action->Equals("E")) {
    cranePillar_TargetX = crane_pillar_Translate[0] + 4.0f;
    cranePillarMoving = true;
    shouldReleaseObject = true;
    cranePillarTimer->Start();
}
else if (action->Equals("Q")) {
```

```
            cranePillar_TargetX = crane_pillar_Translate[0] - 4.0f;
            cranePillarMoving = true;
            shouldReleaseObject = true;
            cranePillarTimer->Start();
        }
        else if (action->Equals("R")) {
            object_Pos[0] = 0.0f;
            object_Pos[1] = 1.35f;
            object_Pos[2] = -7.0f;
            cubeColor[0] = 1.0f;
            cubeColor[1] = 0.0f;
            cubeColor[2] = 0.0f;
            cubeColor[3] = 0.5f;
            shouldReleaseObject = false;
            objectAttached = false;
            clawPauseCounter = 0;
            finishedPartsCount++;
            MessageBox::Show(String::Format("Кількість готових деталей: {0}", finishedPartsCount),
                "Готові деталі",
                MessageBoxButtons::OK,
                MessageBoxIcon::None);
        }
        playbackIndex++;
        this->RenderScene(nullptr, nullptr);
    }

    void MyForm_KeyDown(Object^ sender, KeyEventArgs^ e) {
        if (e->KeyCode == Keys::D1) {
            crane_TargetZ = -7.0f;
```

```
    craneMoving = true;
    craneTimer->Start();
    recordAction("1");
}
if (e->KeyCode == Keys::D2) {
    crane_TargetZ = -2.0f;
    shouldReleaseObject = true;
    craneMoving = true;
    craneTimer->Start();
    recordAction("2");
}
if (e->KeyCode == Keys::D3) {
    crane_TargetZ = 3.0f;
    shouldReleaseObject = true;
    craneMoving = true;
    craneTimer->Start();
    recordAction("3");
}
if (e->KeyCode == Keys::D4) {
    crane_TargetZ = 7.0f;
    shouldReleaseObject = true;
    craneMoving = true;
    craneTimer->Start();
    recordAction("4");
}

if (e->KeyCode == Keys::B) {
    if (objectAttached) {
```

```
                MessageBox::Show("Не вдалося запустити анімацію верстата. Об'єкт все ще прикріплений до крана. Спочатку відпустіть об'єкт.", "Інформація", MessageBoxButtons::OK, MessageBoxIcon::Warning);
                return;
            }
            MachineInstance^ targetMachine = FindMachineWithObject();
            if (targetMachine != nullptr && targetMachine->isStanok1 && !targetMachine->isDoorAnimationActive) {
                targetMachine->isDoorAnimationActive = true;
                targetMachine->doorPhase = 0;
                targetMachine->pauseCounter = 0;
                doorTimer->Start();
                recordAction("A");
            }
            else {
                MessageBox::Show("Не вдалося запустити анімацію дверей. Об'єкт не на верстаті типу 'Верстат 1' або анімація вже активна.", "Інформація");
            }
        }

        if (e->KeyCode == Keys::N) {
            if (objectAttached) {
                MessageBox::Show("Не вдалося запустити анімацію верстата. Об'єкт все ще прикріплений до крана. Спочатку відпустіть об'єкт.", "Інформація", MessageBoxButtons::OK, MessageBoxIcon::Warning);
                return;
            }
            MachineInstance^ targetMachine = FindMachineWithObject();
            if (targetMachine != nullptr && !targetMachine->isStanok1 && !targetMachine->spindleAnimationActive) {
```

```
            targetMachine->spindleAnimationActive = true;
            targetMachine->spindlePhase = 0;
            targetMachine->spindleAngle = 0.0f;
            spindleTimer->Start();
            recordAction("B");
        }
        else {
            MessageBox::Show("Не вдалося запустити анімацію шпинделя. Об'єкт не на верстаті типу 'Верстат 2' або анімація вже активна.", "Інформація");
        }
    }
    if (e->KeyCode == Keys::V) {
        if (!clawsMoving) {
            clawsMoving = true;
            clawsPhase = 1;
            clawsTimer->Start();
            recordAction("C");
        }
    }
    if (e->KeyCode == Keys::R) {
        object_Pos[0] = 0.0f;
        object_Pos[1] = 1.35f;
        object_Pos[2] = -7.0f;

        cubeColor[0] = 1.0f;
        cubeColor[1] = 0.0f;
        cubeColor[2] = 0.0f;
        cubeColor[3] = 0.5f;
        shouldReleaseObject = false;
        objectAttached = false;
```

```
            finishedPartsCount++;
            recordAction("R");
            MessageBox::Show(String::Format("Кількість готових деталей: {0}", finishedPartsCount),
                "Готові деталі",
                MessageBoxButtons::OK,
                MessageBoxIcon::None);
        }
        if (e->KeyCode == Keys::T) {
            object_Pos[0] = 0.0f;
            object_Pos[1] = 1.35f;
            object_Pos[2] = -7.0f;

            cubeColor[0] = 1.0f;
            cubeColor[1] = 0.0f;
            cubeColor[2] = 0.0f;
            cubeColor[3] = 0.5f;
            shouldReleaseObject = false;
            objectAttached = false;
        }
        if (e->KeyCode == Keys::E) {
            cranePillar_TargetX = crane_pillar_Translate[0] + 4.0f;
            cranePillarMoving = true;
            shouldReleaseObject = true;
            cranePillarTimer->Start();
            recordAction("E");
        }
        if (e->KeyCode == Keys::Q) {
            cranePillar_TargetX = crane_pillar_Translate[0] - 4.0f;
```

```
    cranePillarMoving = true;
    shouldReleaseObject = true;
    cranePillarTimer->Start();
    recordAction("Q");
}
if (e->KeyCode == Keys::W) {
    if (!isRecording) {
        isRecording = true;
        recordedActions->Clear();
        MessageBox::Show("Запис розпочато");
    }
    else {
        isRecording = false;
        MessageBox::Show("Запис зупинено");
    }
}
if (e->KeyCode == Keys::S) {
    if (recordedActions->Count > 0 && !isPlaying) {
        isPlaying = true;
        playbackIndex = 0;
        lastPlaybackTime = System::DateTime::Now;
        this->countTimer->Start();
        MessageBox::Show("Відтворення розпочато");
    }
    else {
        isPlaying = false;
        this->countTimer->Stop();
        MessageBox::Show("Відтворення закінчено");
    }
}
```

```
            this->RenderScene(nullptr, nullptr);
        }
        void MyForm_Load(System::Object^ sender, System::EventArgs^ e)
        {
            glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
            SetupProjection(this->ClientSize.Width, this->ClientSize.Height);
        }
        void MyForm_Resize(System::Object^ sender, System::EventArgs^ e)
        {
            SetupProjection(this->ClientSize.Width, this->ClientSize.Height);
        }
        void MyForm_Closing(System::Object^ sender,
System::Windows::Forms::FormClosingEventArgs^ e)
        {
            renderTimer->Stop();
            wglMakeCurrent(NULL, NULL);
            wglDeleteContext(hglrc);
            ReleaseDC((HWND)this->Handle.ToPointer(), hdc);
        }
        void vScrollBar1_Scroll(System::Object^ sender,
System::Windows::Forms::ScrollEventArgs^ e) {
            vspos = e->NewValue;
            this->RenderScene(nullptr, nullptr);
        }
        void hScrollBar1_Scroll(System::Object^ sender,
System::Windows::Forms::ScrollEventArgs^ e) {
            hspos = e->NewValue;
            this->RenderScene(nullptr, nullptr);
        }
```

```
        void MyForm_MouseWheel(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e)
        {
            if (e->Delta > 0)
                zoom -= 0.5f;
            else
                zoom += 0.5f;
            if (zoom < 1.0f) zoom = 1.0f;
            if (zoom > 50.0f) zoom = 50.0f;

            this->RenderScene(nullptr, nullptr);
        }
        void btnDeleteLast_Click(System::Object^ sender, System::EventArgs^ e)
{
            if (placedMachines->Count > 0) {
                placedMachines->RemoveAt(placedMachines->Count - 1);
                if (nextMachinePositionIndex > 0) {
                    nextMachinePositionIndex--;
                }
                this->RenderScene(nullptr, nullptr);
            }
            else {
                MessageBox::Show("Немає верстатів для видалення.",
"Інформація");
            }
        }
        void btnDeleteAll_Click(System::Object^ sender, System::EventArgs^ e) {
            if (placedMachines->Count > 0) {
```

```
            if (MessageBox::Show("Ви впевнені, що хочете видалити всі верстати?", "Підтвердження", MessageBoxButtons::YesNo, MessageBoxIcon::Question) == System::Windows::Forms::DialogResult::Yes) {
                placedMachines->Clear();
                nextMachinePositionIndex = 0;
                this->RenderScene(nullptr, nullptr);
            }
        }
        else {
            MessageBox::Show("Немає верстатів для видалення.", "Інформація");
        }
    }
    void btnDeleteCount_Click(System::Object^ sender, System::EventArgs^ e) {
        finishedPartsCount = 0;
        MessageBox::Show(String::Format("Кількість готових деталей: {0}", finishedPartsCount),
            "Готові деталі",
            MessageBoxButtons::OK,
            MessageBoxIcon::None);
    }
    void btnListCommands_Click(System::Object^ sender, System::EventArgs^ e) {
        String^ commandsList =
            "1. 1,2,3,4 - рух між позиціями верстатів;\n" +
            "2. E,Q - рух кран балки;\n" +
            "3. R - зброс положення об'єкта;\n" +
            "4. W - початок/кінець запису дій;\n" +
            "5. S - початок/кінець відтворення дій;\n" +
```

```
            "6. V - опускання/підняття об'єкта\n" +
            "7. B - запуск анімації для першого типу верстатів\n" +
            "8. N - запуск анімації для другого типу верстатів";

        MessageBox::Show(commandsList,
            "Список керуючих команд",
            MessageBoxButtons::OK,
            MessageBoxIcon::None);
    }
    void btnAddStanok_Click(System::Object^ sender, System::EventArgs^ e)
{
        DD::MachineQuantityDialog^ quantityDialog = gcnew
DD::MachineQuantityDialog();

        int requestedQuantity = 0;
        bool selectedStanokType = true;

        if (quantityDialog->ShowDialog() ==
System::Windows::Forms::DialogResult::OK) {
            requestedQuantity = quantityDialog->SelectedQuantity;
            selectedStanokType = quantityDialog->IsStanok1Selected;

            int addedCount = 0;
            for (int i = 0; i < requestedQuantity; ++i) {
                if (nextMachinePositionIndex < predefinedPositions->Length) {
                    Tuple<GLfloat, GLfloat, GLfloat>^ pos =
predefinedPositions[nextMachinePositionIndex];
                    placedMachines->Add(gcnew MachineInstance(pos->Item1,
pos->Item2, pos->Item3, selectedStanokType));
                    nextMachinePositionIndex++;
```

```
                addedCount++;
            }
            else {
                MessageBox::Show("Досягнуто максимальної кількості верстатів, які можна розмістити на заданих позиціях.", "Обмеження", MessageBoxButtons::OK, MessageBoxIcon::Information);
                break;
            }
        }

        if (addedCount > 0) {
            MessageBox::Show(String::Format("Додано {0} верстатів типу '{1}'.",
                addedCount,
                selectedStanokType ? "Верстат 1" : "Верстат 2"), "Інформація");
            this->RenderScene(nullptr, nullptr);
        }
        else if (requestedQuantity > 0) {
            MessageBox::Show("Не вдалося додати жодного верстата, всі задані позиції зайняті.", "Помилка", MessageBoxButtons::OK, MessageBoxIcon::Warning);
        }
    }
    delete quantityDialog;
    this->RenderScene(nullptr, nullptr);
}
void btnSettingsButton_Click(System::Object^ sender, System::EventArgs^ e) {
    DD::SettingsForm^ settingsDialog = gcnew DD::SettingsForm(
```

```
                this->pauseLimit,
                this->door_scale_change,
                this->move_speed,
                this->machine1Stability,
                this->machine1Resource,
                this->machine2Stability,
                this->machine2Resource
            );
            if (settingsDialog->ShowDialog(this) == System::Windows::Forms::DialogResult::OK) {
                this->pauseLimit = settingsDialog->ResultPauseLimit;
                this->door_scale_change = settingsDialog->ResultDoorScaleChange;
                this->move_speed = settingsDialog->ResultMoveSpeed;
                this->machine1Stability = settingsDialog->ResultMachine1Stability;
                this->machine1Resource = settingsDialog->ResultMachine1Resource;
                this->machine2Stability = settingsDialog->ResultMachine2Stability;
                this->machine2Resource = settingsDialog->ResultMachine2Resource;

                MessageBox::Show("Налаштування верстатів оновлено!", "Інформація");
                this->RenderScene(nullptr, nullptr);
            }
            delete settingsDialog;
        }
    };

}
```

ДОДАТОК Г

Код допоміжної форми «MachineQuantityDialog» Visual Studio

```
#pragma once

namespace DD {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    public ref class MachineQuantityDialog : public
System::Windows::Forms::Form
    {
    public:
      MachineQuantityDialog(void)
      {
        InitializeComponent();
        this->DialogResult = System::Windows::Forms::DialogResult::Cancel;
        radioButtonStanok1->Checked = true;
      }

    protected:
      ~MachineQuantityDialog()
      {
        if (components)
```

```
            {
                delete components;
            }
        }

    private:
        System::Windows::Forms::NumericUpDown^ numericUpDown1;
        System::Windows::Forms::Button^ okButton;
        System::Windows::Forms::Label^ label1;
        System::Windows::Forms::GroupBox^ groupBoxMachineType;
        System::Windows::Forms::RadioButton^ radioButtonStanok1;
        System::Windows::Forms::RadioButton^ radioButtonStanok2;

        System::ComponentModel::Container^ components;

    #pragma region Windows Form Designer generated code
        void InitializeComponent(void)
        {
            this->components = gcnew System::ComponentModel::Container();

            this->numericUpDown1 = (gcnew
System::Windows::Forms::NumericUpDown());
            this->okButton = (gcnew System::Windows::Forms::Button());
            this->label1 = (gcnew System::Windows::Forms::Label());
            this->groupBoxMachineType = (gcnew
System::Windows::Forms::GroupBox());
            this->radioButtonStanok1 = (gcnew
System::Windows::Forms::RadioButton());
            this->radioButtonStanok2 = (gcnew
System::Windows::Forms::RadioButton());
```

```
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))->BeginInit();

        this->SuspendLayout();
        this->Size = System::Drawing::Size(300, 250);
        this->Text = L"Вибір верстатів";
        this->FormBorderStyle = System::Windows::Forms::FormBorderStyle::FixedDialog;
        this->MaximizeBox = false;
        this->MinimizeBox = false;
        this->StartPosition = System::Windows::Forms::FormStartPosition::CenterParent;
        this->Padding = System::Windows::Forms::Padding(0);

        this->label1->AutoSize = true;
        this->label1->Location = System::Drawing::Point(20, 30);
        this->label1->Name = L"label1";
        this->label1->Size = System::Drawing::Size(120, 16);
        this->label1->TabIndex = 2;
        this->label1->Text = L"Кількість верстатів (1-10):";

        this->numericUpDown1->Location = System::Drawing::Point(175, 28);
        this->numericUpDown1->Maximum = System::Decimal(10);
        this->numericUpDown1->Minimum = System::Decimal(1);
        this->numericUpDown1->Name = L"numericUpDown1";
        this->numericUpDown1->Size = System::Drawing::Size(60, 22);
        this->numericUpDown1->TabIndex = 0;
        this->numericUpDown1->Value = System::Decimal(1);
```

```
            this->groupBoxMachineType->Location = System::Drawing::Point(20,
70);
            this->groupBoxMachineType->Name = L"groupBoxMachineType";
            this->groupBoxMachineType->Size = System::Drawing::Size(250, 70);
            this->groupBoxMachineType->TabIndex = 3;
            this->groupBoxMachineType->TabStop = false;
            this->groupBoxMachineType->Text = L"Тип верстата";

            this->radioButtonStanok1->AutoSize = true;
            this->radioButtonStanok1->Location = System::Drawing::Point(20, 20);
            this->radioButtonStanok1->Name = L"radioButtonStanok1";
            this->radioButtonStanok1->Size = System::Drawing::Size(85, 20);
            this->radioButtonStanok1->TabIndex = 0;
            this->radioButtonStanok1->TabStop = true;
            this->radioButtonStanok1->Text = L"Верстат 1";
            this->radioButtonStanok1->UseVisualStyleBackColor = true;

            this->radioButtonStanok2->AutoSize = true;
            this->radioButtonStanok2->Location = System::Drawing::Point(140,
20);
            this->radioButtonStanok2->Name = L"radioButtonStanok2";
            this->radioButtonStanok2->Size = System::Drawing::Size(85, 20);
            this->radioButtonStanok2->TabIndex = 1;
            this->radioButtonStanok2->Text = L"Верстат 2";
            this->radioButtonStanok2->UseVisualStyleBackColor = true;

            this->groupBoxMachineType->Controls->Add(this-
>radioButtonStanok2);
            this->groupBoxMachineType->Controls->Add(this-
>radioButtonStanok1);
```

```
            this->okButton->Location = System::Drawing::Point(105, 160);
            this->okButton->Name = L"okButton";
            this->okButton->Size = System::Drawing::Size(75, 25);
            this->okButton->TabIndex = 1;
            this->okButton->Text = L"OK";
            this->okButton->UseVisualStyleBackColor = true;
            this->okButton->Click += gcnew System::EventHandler(this,
&MachineQuantityDialog::okButton_Click);

            this->Controls->Add(this->label1);
            this->Controls->Add(this->okButton);
            this->Controls->Add(this->numericUpDown1);
            this->Controls->Add(this->groupBoxMachineType);

            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>numericUpDown1))->EndInit();
            this->ResumeLayout(false);
            this->PerformLayout();
        }
    #pragma endregion

        System::Void okButton_Click(System::Object^ sender,
System::EventArgs^ e)
        {
            this->DialogResult = System::Windows::Forms::DialogResult::OK;
            this->Close();
        }
    public:
        property int SelectedQuantity{
```

```
            int get() {
                return (int)numericUpDown1->Value;
            }
        }
            property bool IsStanok1Selected{
                bool get() {
                    return radioButtonStanok1->Checked;
                }
        }
    };
}
```

ДОДАТОК Д

Код допоміжної форми «SettingsForm» Visual Studio

```
#pragma once
#include <cmath>

namespace DD {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    public ref class SettingsForm : public System::Windows::Forms::Form
    {
    public:
        property int ResultPauseLimit;
        property float ResultDoorScaleChange;
        property float ResultMoveSpeed;
        property int ResultMachine1Stability;
        property int ResultMachine1Resource;
        property int ResultMachine2Stability;
        property int ResultMachine2Resource;

        SettingsForm(int initialPauseLimit, float initialDoorScale, float initialMoveSpeed,
            int initialM1Stability, int initialM1Resource,
```

```
        int initialM2Stability, int initialM2Resource)
    {
        InitializeComponent();
        this->currentInitialPauseLimit = initialPauseLimit;
        this->currentInitialDoorScaleChange = initialDoorScale;
        this->currentInitialMoveSpeed = initialMoveSpeed;
        this->currentInitialMachine1Stability = initialM1Stability;
        this->currentInitialMachine1Resource = initialM1Resource;
        this->currentInitialMachine2Stability = initialM2Stability;
        this->currentInitialMachine2Resource = initialM2Resource;
        SetInitialTrackBarValues();
        UpdateAllValueLabels();
    }

protected:
    ~SettingsForm()
    {
        if (components)
        {
            delete components;
        }
    }

private:
    int currentInitialPauseLimit;
    float currentInitialDoorScaleChange;
    float currentInitialMoveSpeed;
    int currentInitialMachine1Stability;
    int currentInitialMachine1Resource;
    int currentInitialMachine2Stability;
```

```
int currentInitialMachine2Resource;
System::Windows::Forms::GroupBox^ groupBoxMachine1;
System::Windows::Forms::Label^ labelDoorSpeedPrompt;
System::Windows::Forms::TrackBar^ trackBarDoorSpeed;
System::Windows::Forms::Label^ labelDoorSpeedValue;
System::Windows::Forms::Label^ labelProcessingDelayPrompt;
System::Windows::Forms::TrackBar^ trackBarProcessingDelay;
System::Windows::Forms::Label^ labelProcessingDelayValue;
System::Windows::Forms::GroupBox^ groupBoxMachine2;
System::Windows::Forms::Label^ labelMoveSpeedPrompt;
System::Windows::Forms::TrackBar^ trackBarMoveSpeed;
System::Windows::Forms::Label^ labelMoveSpeedValue;
System::Windows::Forms::Button^ buttonOK;
System::Windows::Forms::Button^ buttonCancel;
System::ComponentModel::Container^ components;
System::Windows::Forms::Label^ labelMachine1StabilityPrompt;
System::Windows::Forms::TrackBar^ trackBarMachine1Stability;
System::Windows::Forms::Label^ labelMachine1StabilityValue;
System::Windows::Forms::Label^ labelMachine1ResourcePrompt;
System::Windows::Forms::TrackBar^ trackBarMachine1Resource;
System::Windows::Forms::Label^ labelMachine1ResourceValue;

System::Windows::Forms::Label^ labelMachine2StabilityPrompt;
System::Windows::Forms::TrackBar^ trackBarMachine2Stability;
System::Windows::Forms::Label^ labelMachine2StabilityValue;
System::Windows::Forms::Label^ labelMachine2ResourcePrompt;
System::Windows::Forms::TrackBar^ trackBarMachine2Resource;
System::Windows::Forms::Label^ labelMachine2ResourceValue;
```

```
#pragma region Windows Form Designer generated code
        void InitializeComponent(void)
        {
            this->components = (gcnew System::ComponentModel::Container());

            this->groupBoxMachine1 = (gcnew
System::Windows::Forms::GroupBox());
            this->labelDoorSpeedPrompt = (gcnew
System::Windows::Forms::Label());
            this->trackBarDoorSpeed = (gcnew
System::Windows::Forms::TrackBar());
            this->labelDoorSpeedValue = (gcnew
System::Windows::Forms::Label());
            this->labelProcessingDelayPrompt = (gcnew
System::Windows::Forms::Label());
            this->trackBarProcessingDelay = (gcnew
System::Windows::Forms::TrackBar());
            this->labelProcessingDelayValue = (gcnew
System::Windows::Forms::Label());
            this->labelMachine1StabilityPrompt = (gcnew
System::Windows::Forms::Label());
            this->trackBarMachine1Stability = (gcnew
System::Windows::Forms::TrackBar());
            this->labelMachine1StabilityValue = (gcnew
System::Windows::Forms::Label());
            this->labelMachine1ResourcePrompt = (gcnew
System::Windows::Forms::Label());
            this->trackBarMachine1Resource = (gcnew
System::Windows::Forms::TrackBar());
```

```
            this->labelMachine1ResourceValue = (gcnew
System::Windows::Forms::Label());
            this->groupBoxMachine2 = (gcnew
System::Windows::Forms::GroupBox());
            this->labelMoveSpeedPrompt = (gcnew
System::Windows::Forms::Label());
            this->trackBarMoveSpeed = (gcnew
System::Windows::Forms::TrackBar());
            this->labelMoveSpeedValue = (gcnew
System::Windows::Forms::Label());
            this->labelMachine2StabilityPrompt = (gcnew
System::Windows::Forms::Label());
            this->trackBarMachine2Stability = (gcnew
System::Windows::Forms::TrackBar());
            this->labelMachine2StabilityValue = (gcnew
System::Windows::Forms::Label());
            this->labelMachine2ResourcePrompt = (gcnew
System::Windows::Forms::Label());
            this->trackBarMachine2Resource = (gcnew
System::Windows::Forms::TrackBar());
            this->labelMachine2ResourceValue = (gcnew
System::Windows::Forms::Label());

            this->buttonOK = (gcnew System::Windows::Forms::Button());
            this->buttonCancel = (gcnew System::Windows::Forms::Button());

            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>trackBarDoorSpeed))->BeginInit();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>trackBarProcessingDelay))->BeginInit();
```

```
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->trackBarMoveSpeed))->BeginInit();

            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->trackBarMachine1Stability))->BeginInit();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->trackBarMachine1Resource))->BeginInit();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->trackBarMachine2Stability))->BeginInit();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->trackBarMachine2Resource))->BeginInit();

            this->groupBoxMachine1->SuspendLayout();
            this->groupBoxMachine2->SuspendLayout();
            this->SuspendLayout();

            this->groupBoxMachine1->Controls->Add(this->labelDoorSpeedValue);
            this->groupBoxMachine1->Controls->Add(this->trackBarDoorSpeed);
            this->groupBoxMachine1->Controls->Add(this->labelDoorSpeedPrompt);
            this->groupBoxMachine1->Controls->Add(this->labelProcessingDelayValue);
            this->groupBoxMachine1->Controls->Add(this->trackBarProcessingDelay);
            this->groupBoxMachine1->Controls->Add(this->labelProcessingDelayPrompt);
            this->groupBoxMachine1->Controls->Add(this->labelMachine1StabilityValue);
```

```
            this->groupBoxMachine1->Controls->Add(this-
>trackBarMachine1Stability);
            this->groupBoxMachine1->Controls->Add(this-
>labelMachine1StabilityPrompt);
            this->groupBoxMachine1->Controls->Add(this-
>labelMachine1ResourceValue);
            this->groupBoxMachine1->Controls->Add(this-
>trackBarMachine1Resource);
            this->groupBoxMachine1->Controls->Add(this-
>labelMachine1ResourcePrompt);

            this->groupBoxMachine1->Location = System::Drawing::Point(12, 12);
            this->groupBoxMachine1->Name = L"groupBoxMachine1";
            this->groupBoxMachine1->Size = System::Drawing::Size(360, 230);
            this->groupBoxMachine1->TabIndex = 0;
            this->groupBoxMachine1->TabStop = false;
            this->groupBoxMachine1->Text = L"Перший верстат";

            this->labelDoorSpeedPrompt->AutoSize = true;
            this->labelDoorSpeedPrompt->Location = System::Drawing::Point(15,
30);
            this->labelDoorSpeedPrompt->Name = L"labelDoorSpeedPrompt";
            this->labelDoorSpeedPrompt->Size = System::Drawing::Size(140, 13);
            this->labelDoorSpeedPrompt->TabIndex = 0;
            this->labelDoorSpeedPrompt->Text = L"Швидкість руху дверей:";

            this->trackBarDoorSpeed->Location = System::Drawing::Point(160,
25);
            this->trackBarDoorSpeed->Maximum = 100;
            this->trackBarDoorSpeed->Minimum = 1;
```

```
            this->trackBarDoorSpeed->Name = L"trackBarDoorSpeed";
            this->trackBarDoorSpeed->Size = System::Drawing::Size(150, 45);
            this->trackBarDoorSpeed->TabIndex = 1;
            this->trackBarDoorSpeed->TickFrequency = 10;
            this->trackBarDoorSpeed->Value = 1;
            this->trackBarDoorSpeed->Scroll += gcnew
System::EventHandler(this, &SettingsForm::trackBarDoorSpeed_Scroll);

            this->labelDoorSpeedValue->AutoSize = true;
            this->labelDoorSpeedValue->Location = System::Drawing::Point(315,
30);
            this->labelDoorSpeedValue->Name = L"labelDoorSpeedValue";
            this->labelDoorSpeedValue->Size = System::Drawing::Size(34, 13);
            this->labelDoorSpeedValue->TabIndex = 2;
            this->labelDoorSpeedValue->Text = L"0.01";

            this->labelProcessingDelayPrompt->AutoSize = true;
            this->labelProcessingDelayPrompt->Location =
System::Drawing::Point(15, 80);
            this->labelProcessingDelayPrompt->Name =
L"labelProcessingDelayPrompt";
            this->labelProcessingDelayPrompt->Size = System::Drawing::Size(145,
13);
            this->labelProcessingDelayPrompt->TabIndex = 3;
            this->labelProcessingDelayPrompt->Text = L"Швидкість обробки
деталі:";

            this->trackBarProcessingDelay->Location =
System::Drawing::Point(160, 75);
            this->trackBarProcessingDelay->Maximum = 100;
```

```
            this->trackBarProcessingDelay->Minimum = 1;
            this->trackBarProcessingDelay->Name = L"trackBarProcessingDelay";
            this->trackBarProcessingDelay->Size = System::Drawing::Size(150,
45);
            this->trackBarProcessingDelay->TabIndex = 4;
            this->trackBarProcessingDelay->TickFrequency = 10;
            this->trackBarProcessingDelay->Value = 1;
            this->trackBarProcessingDelay->Scroll += gcnew
System::EventHandler(this, &SettingsForm::trackBarProcessingDelay_Scroll);

            this->labelProcessingDelayValue->AutoSize = true;
            this->labelProcessingDelayValue->Location =
System::Drawing::Point(315, 80);
            this->labelProcessingDelayValue->Name =
L"labelProcessingDelayValue";
            this->labelProcessingDelayValue->Size = System::Drawing::Size(34,
13);
            this->labelProcessingDelayValue->TabIndex = 5;
            this->labelProcessingDelayValue->Text = L"1000";


            this->labelMachine1StabilityPrompt->AutoSize = true;
            this->labelMachine1StabilityPrompt->Location =
System::Drawing::Point(15, 130);
            this->labelMachine1StabilityPrompt->Name =
L"labelMachine1StabilityPrompt";
            this->labelMachine1StabilityPrompt->Size =
System::Drawing::Size(100, 13);
            this->labelMachine1StabilityPrompt->TabIndex = 6;
            this->labelMachine1StabilityPrompt->Text = L"Стабільність:";
```

```
            this->trackBarMachine1Stability->Location = System::Drawing::Point(160, 125);
            this->trackBarMachine1Stability->Maximum = 100;
            this->trackBarMachine1Stability->Minimum = 1;
            this->trackBarMachine1Stability->Name = L"trackBarMachine1Stability";
            this->trackBarMachine1Stability->Size = System::Drawing::Size(150, 45);
            this->trackBarMachine1Stability->TabIndex = 7;
            this->trackBarMachine1Stability->TickFrequency = 10;
            this->trackBarMachine1Stability->Value = 1;
            this->trackBarMachine1Stability->Scroll += gcnew System::EventHandler(this, &SettingsForm::trackBarMachine1Stability_Scroll);

            this->labelMachine1StabilityValue->AutoSize = true;
            this->labelMachine1StabilityValue->Location = System::Drawing::Point(315, 130);
            this->labelMachine1StabilityValue->Name = L"labelMachine1StabilityValue";
            this->labelMachine1StabilityValue->Size = System::Drawing::Size(25, 13);
            this->labelMachine1StabilityValue->TabIndex = 8;
            this->labelMachine1StabilityValue->Text = L"1";

            this->labelMachine1ResourcePrompt->AutoSize = true;
            this->labelMachine1ResourcePrompt->Location = System::Drawing::Point(15, 180);
            this->labelMachine1ResourcePrompt->Name = L"labelMachine1ResourcePrompt";
```

```
            this->labelMachine1ResourcePrompt->Size =
System::Drawing::Size(50, 13);
            this->labelMachine1ResourcePrompt->TabIndex = 9;
            this->labelMachine1ResourcePrompt->Text = L"Ресурс:";

            this->trackBarMachine1Resource->Location =
System::Drawing::Point(160, 175);
            this->trackBarMachine1Resource->Maximum = 100;
            this->trackBarMachine1Resource->Minimum = 1;
            this->trackBarMachine1Resource->Name =
L"trackBarMachine1Resource";
            this->trackBarMachine1Resource->Size = System::Drawing::Size(150,
45);
            this->trackBarMachine1Resource->TabIndex = 10;
            this->trackBarMachine1Resource->TickFrequency = 10;
            this->trackBarMachine1Resource->Value = 1;
            this->trackBarMachine1Resource->Scroll += gcnew
System::EventHandler(this, &SettingsForm::trackBarMachine1Resource_Scroll);

            this->labelMachine1ResourceValue->AutoSize = true;
            this->labelMachine1ResourceValue->Location =
System::Drawing::Point(315, 180);
            this->labelMachine1ResourceValue->Name =
L"labelMachine1ResourceValue";
            this->labelMachine1ResourceValue->Size = System::Drawing::Size(25,
13);
            this->labelMachine1ResourceValue->TabIndex = 11;
            this->labelMachine1ResourceValue->Text = L"1";
```

```
            this->groupBoxMachine2->Controls->Add(this->labelMoveSpeedValue);
            this->groupBoxMachine2->Controls->Add(this->trackBarMoveSpeed);
            this->groupBoxMachine2->Controls->Add(this->labelMoveSpeedPrompt);
            this->groupBoxMachine2->Controls->Add(this->labelMachine2StabilityValue);
            this->groupBoxMachine2->Controls->Add(this->trackBarMachine2Stability);
            this->groupBoxMachine2->Controls->Add(this->labelMachine2StabilityPrompt);
            this->groupBoxMachine2->Controls->Add(this->labelMachine2ResourceValue);
            this->groupBoxMachine2->Controls->Add(this->trackBarMachine2Resource);
            this->groupBoxMachine2->Controls->Add(this->labelMachine2ResourcePrompt);

            this->groupBoxMachine2->Location = System::Drawing::Point(12, 255);
            this->groupBoxMachine2->Name = L"groupBoxMachine2";
            this->groupBoxMachine2->Size = System::Drawing::Size(360, 180);
            this->groupBoxMachine2->TabIndex = 1;
            this->groupBoxMachine2->TabStop = false;
            this->groupBoxMachine2->Text = L"Другий верстат";

            this->labelMoveSpeedPrompt->AutoSize = true;
            this->labelMoveSpeedPrompt->Location = System::Drawing::Point(15, 30);
            this->labelMoveSpeedPrompt->Name = L"labelMoveSpeedPrompt";
```

```
            this->labelMoveSpeedPrompt->Size = System::Drawing::Size(90, 13);
            this->labelMoveSpeedPrompt->TabIndex = 0;
            this->labelMoveSpeedPrompt->Text = L"Швидкість руху:";

            this->trackBarMoveSpeed->Location = System::Drawing::Point(160,
25);
            this->trackBarMoveSpeed->Maximum = 100;
            this->trackBarMoveSpeed->Minimum = 1;
            this->trackBarMoveSpeed->Name = L"trackBarMoveSpeed";
            this->trackBarMoveSpeed->Size = System::Drawing::Size(150, 45);
            this->trackBarMoveSpeed->TabIndex = 1;
            this->trackBarMoveSpeed->TickFrequency = 10;
            this->trackBarMoveSpeed->Value = 1;
            this->trackBarMoveSpeed->Scroll += gcnew
System::EventHandler(this, &SettingsForm::trackBarMoveSpeed_Scroll);

            this->labelMoveSpeedValue->AutoSize = true;
            this->labelMoveSpeedValue->Location = System::Drawing::Point(315,
30);
            this->labelMoveSpeedValue->Name = L"labelMoveSpeedValue";
            this->labelMoveSpeedValue->Size = System::Drawing::Size(34, 13);
            this->labelMoveSpeedValue->TabIndex = 2;
            this->labelMoveSpeedValue->Text = L"0.01";

            this->labelMachine2StabilityPrompt->AutoSize = true;
            this->labelMachine2StabilityPrompt->Location =
System::Drawing::Point(15, 80);
            this->labelMachine2StabilityPrompt->Name =
L"labelMachine2StabilityPrompt";
```

```
            this->labelMachine2StabilityPrompt->Size =
System::Drawing::Size(100, 13);
            this->labelMachine2StabilityPrompt->TabIndex = 3;
            this->labelMachine2StabilityPrompt->Text = L"Стабільність:";

            this->trackBarMachine2Stability->Location =
System::Drawing::Point(160, 75);
            this->trackBarMachine2Stability->Maximum = 100;
            this->trackBarMachine2Stability->Minimum = 1;
            this->trackBarMachine2Stability->Name =
L"trackBarMachine2Stability";
            this->trackBarMachine2Stability->Size = System::Drawing::Size(150,
45);
            this->trackBarMachine2Stability->TabIndex = 4;
            this->trackBarMachine2Stability->TickFrequency = 10;
            this->trackBarMachine2Stability->Value = 1;
            this->trackBarMachine2Stability->Scroll += gcnew
System::EventHandler(this, &SettingsForm::trackBarMachine2Stability_Scroll);

            this->labelMachine2StabilityValue->AutoSize = true;
            this->labelMachine2StabilityValue->Location =
System::Drawing::Point(315, 80);
            this->labelMachine2StabilityValue->Name =
L"labelMachine2StabilityValue";
            this->labelMachine2StabilityValue->Size = System::Drawing::Size(25,
13);
            this->labelMachine2StabilityValue->TabIndex = 5;
            this->labelMachine2StabilityValue->Text = L"1";

            this->labelMachine2ResourcePrompt->AutoSize = true;
```

```
this->labelMachine2ResourcePrompt->Location =
System::Drawing::Point(15, 130);
this->labelMachine2ResourcePrompt->Name =
L"labelMachine2ResourcePrompt";
this->labelMachine2ResourcePrompt->Size =
System::Drawing::Size(50, 13);
this->labelMachine2ResourcePrompt->TabIndex = 6;
this->labelMachine2ResourcePrompt->Text = L"Ресурс:";

this->trackBarMachine2Resource->Location =
System::Drawing::Point(160, 125);
this->trackBarMachine2Resource->Maximum = 100;
this->trackBarMachine2Resource->Minimum = 1;
this->trackBarMachine2Resource->Name =
L"trackBarMachine2Resource";
this->trackBarMachine2Resource->Size = System::Drawing::Size(150,
45);
this->trackBarMachine2Resource->TabIndex = 7;
this->trackBarMachine2Resource->TickFrequency = 10;
this->trackBarMachine2Resource->Value = 1;
this->trackBarMachine2Resource->Scroll += gcnew
System::EventHandler(this, &SettingsForm::trackBarMachine2Resource_Scroll);

this->labelMachine2ResourceValue->AutoSize = true;
this->labelMachine2ResourceValue->Location =
System::Drawing::Point(315, 130);
this->labelMachine2ResourceValue->Name =
L"labelMachine2ResourceValue";
this->labelMachine2ResourceValue->Size = System::Drawing::Size(25,
13);
```

```
            this->labelMachine2ResourceValue->TabIndex = 8;
            this->labelMachine2ResourceValue->Text = L"1";

            this->buttonOK->Location = System::Drawing::Point(216, 450);
            this->buttonOK->Name = L"buttonOK";
            this->buttonOK->Size = System::Drawing::Size(75, 23);
            this->buttonOK->TabIndex = 2;
            this->buttonOK->Text = L"OK";
            this->buttonOK->UseVisualStyleBackColor = true;
            this->buttonOK->Click += gcnew System::EventHandler(this,
&SettingsForm::buttonOK_Click);

            this->buttonCancel->DialogResult =
System::Windows::Forms::DialogResult::Cancel;
            this->buttonCancel->Location = System::Drawing::Point(297, 450);
            this->buttonCancel->Name = L"buttonCancel";
            this->buttonCancel->Size = System::Drawing::Size(75, 23);
            this->buttonCancel->TabIndex = 3;
            this->buttonCancel->Text = L"Скасувати";
            this->buttonCancel->UseVisualStyleBackColor = true;
            this->buttonCancel->Click += gcnew System::EventHandler(this,
&SettingsForm::buttonCancel_Click);

            this->AcceptButton = this->buttonOK;
            this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
            this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
            this->CancelButton = this->buttonCancel;
            this->ClientSize = System::Drawing::Size(384, 490);
            this->Controls->Add(this->buttonCancel);
```

```
            this->Controls->Add(this->buttonOK);
            this->Controls->Add(this->groupBoxMachine2);
            this->Controls->Add(this->groupBoxMachine1);
            this->FormBorderStyle =
System::Windows::Forms::FormBorderStyle::FixedDialog;
            this->MaximizeBox = false;
            this->MinimizeBox = false;
            this->Name = L"SettingsForm";
            this->ShowInTaskbar = false;
            this->StartPosition =
System::Windows::Forms::FormStartPosition::CenterParent;
            this->Text = L"Налаштування верстатів";

            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>trackBarDoorSpeed))->EndInit();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>trackBarProcessingDelay))->EndInit();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>trackBarMoveSpeed))->EndInit();

            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>trackBarMachine1Stability))->EndInit();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>trackBarMachine1Resource))->EndInit();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>trackBarMachine2Stability))->EndInit();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>trackBarMachine2Resource))->EndInit();

            this->groupBoxMachine1->ResumeLayout(false);
```

```
            this->groupBoxMachine1->PerformLayout();
            this->groupBoxMachine2->ResumeLayout(false);
            this->groupBoxMachine2->PerformLayout();
            this->ResumeLayout(false);
        }
    #pragma endregion

        private:
            void SetInitialTrackBarValues()
            {
                float doorScaleMin = 0.01f;
                float doorScaleMax = 0.08f;
                if ((doorScaleMax - doorScaleMin) != 0) {
                    trackBarDoorSpeed->Value = Math::Max(1, Math::Min(100,
(int)Math::Round(1.0f + ((currentInitialDoorScaleChange - doorScaleMin) /
(doorScaleMax - doorScaleMin)) * 99.0f)));
                }
                else {
                    trackBarDoorSpeed->Value = 1;
                }
                int delayMin = 100;
                int delayMax = 1000;
                if ((delayMax - delayMin) != 0) {
                    trackBarProcessingDelay->Value = Math::Max(1, Math::Min(100,
(int)Math::Round(1.0f + ((delayMax - currentInitialPauseLimit) / (float)(delayMax -
delayMin)) * 99.0f)));
                }
                else {
                    trackBarProcessingDelay->Value = 1;
                }
```

```
            float moveSpeedMin = 0.01f;
            float moveSpeedMax = 0.06f;
            if ((moveSpeedMax - moveSpeedMin) != 0) {
                trackBarMoveSpeed->Value = Math::Max(1, Math::Min(100, (int)Math::Round(1.0f + ((currentInitialMoveSpeed - moveSpeedMin) / (moveSpeedMax - moveSpeedMin)) * 99.0f)));
            }
            else {
                trackBarMoveSpeed->Value = 1;
            }

            trackBarMachine1Stability->Value = Math::Max(1, Math::Min(100, currentInitialMachine1Stability));
            trackBarMachine1Resource->Value = Math::Max(1, Math::Min(100, currentInitialMachine1Resource));
            trackBarMachine2Stability->Value = Math::Max(1, Math::Min(100, currentInitialMachine2Stability));
            trackBarMachine2Resource->Value = Math::Max(1, Math::Min(100, currentInitialMachine2Resource));
        }
        void UpdateAllValueLabels()
        {
            trackBarDoorSpeed_Scroll(nullptr, nullptr);
            trackBarProcessingDelay_Scroll(nullptr, nullptr);
            trackBarMoveSpeed_Scroll(nullptr, nullptr);
            trackBarMachine1Stability_Scroll(nullptr, nullptr);
            trackBarMachine1Resource_Scroll(nullptr, nullptr);
            trackBarMachine2Stability_Scroll(nullptr, nullptr);
            trackBarMachine2Resource_Scroll(nullptr, nullptr);
        }
```

```
        System::Void trackBarDoorSpeed_Scroll(System::Object^ sender,
System::EventArgs^ e) {
            int sliderValue = trackBarDoorSpeed->Value;
            float calculatedValue = 0.01f + (sliderValue - 1) * (0.08f - 0.01f) / 99.0f;
            labelDoorSpeedValue->Text = String::Format("{0:F4}",
calculatedValue);
        }
        System::Void trackBarProcessingDelay_Scroll(System::Object^ sender,
System::EventArgs^ e) {
            int sliderValue = trackBarProcessingDelay->Value;
            int calculatedValue = (int)Math::Round(1000.0f - (sliderValue - 1) *
(900.0f / 99.0f));
            labelProcessingDelayValue->Text = calculatedValue.ToString();
        }
        System::Void trackBarMoveSpeed_Scroll(System::Object^ sender,
System::EventArgs^ e) {
            int sliderValue = trackBarMoveSpeed->Value;
            float calculatedValue = 0.01f + (sliderValue - 1) * (0.06f - 0.01f) / 99.0f;
            labelMoveSpeedValue->Text = String::Format("{0:F4}",
calculatedValue);
        }
        System::Void trackBarMachine1Stability_Scroll(System::Object^ sender,
System::EventArgs^ e) {
            labelMachine1StabilityValue->Text = trackBarMachine1Stability-
>Value.ToString();
        }
        System::Void trackBarMachine1Resource_Scroll(System::Object^ sender,
System::EventArgs^ e) {
            labelMachine1ResourceValue->Text = trackBarMachine1Resource-
>Value.ToString();
```

```
        }
        System::Void trackBarMachine2Stability_Scroll(System::Object^ sender,
System::EventArgs^ e) {
            labelMachine2StabilityValue->Text = trackBarMachine2Stability-
>Value.ToString();
        }
        System::Void trackBarMachine2Resource_Scroll(System::Object^ sender,
System::EventArgs^ e) {
            labelMachine2ResourceValue->Text = trackBarMachine2Resource-
>Value.ToString();
        }
        System::Void buttonOK_Click(System::Object^ sender,
System::EventArgs^ e) {
            int sliderDoorSpeed = trackBarDoorSpeed->Value;
            ResultDoorScaleChange = 0.01f + (sliderDoorSpeed - 1) * (0.08f - 0.01f)
/ 99.0f;
            int sliderProcessingDelay = trackBarProcessingDelay->Value;
            ResultPauseLimit = (int)Math::Round(1000.0f - (sliderProcessingDelay
- 1) * (900.0f / 99.0f));
            int sliderMoveSpeed = trackBarMoveSpeed->Value;
            ResultMoveSpeed = 0.01f + (sliderMoveSpeed - 1) * (0.06f - 0.01f) /
99.0f;
            ResultMachine1Stability = trackBarMachine1Stability->Value;
            ResultMachine1Resource = trackBarMachine1Resource->Value;
            ResultMachine2Stability = trackBarMachine2Stability->Value;
            ResultMachine2Resource = trackBarMachine2Resource->Value;
            this->DialogResult = System::Windows::Forms::DialogResult::OK;
            this->Close();
        }
```

```
        System::Void buttonCancel_Click(System::Object^ sender,
System::EventArgs^ e) {
            this->DialogResult = System::Windows::Forms::DialogResult::Cancel;
            this->Close();
        }
    };
}
```

ДОДАТОК Е

Демонстраційний матеріал

№ докумен-та	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1	ГЮИК. 501600.001 ПЗ	Пояснювальна записка	А4, 178 с.
		Додаткові матеріали	
2		Апробація результатів роботи (теза)	А4, 4 с.
3		Апробація результатів роботи (стаття)	А4, 5 с.
4		Код головної форми «MyForm.h» Visual Studio	А4, 63 с.
5		Код допоміжної форми «MachineQuantityDialog.h» Visual	А4, 6 с.
6		Код допоміжної форми «Settingsform.h» Visual Studio	А4, 23 с.
7		Демонстраційний матеріал	А4, 12 с.

Змін.	Арк.	Номер докум.	Підпис	Дата	ГЮИК. 501600.001 ВД		
Розроб.		Берест			Розроблення віртуального макету для дослідження гнучкої виробничої дільниці	Літера	Аркуш	Аркушів
Перевір.		Гурін				Н	1	1
						Кафедра КІТАР ХНУРЕ		
Н.контр.		Стародубцев						
Затв.		Невлюдов						