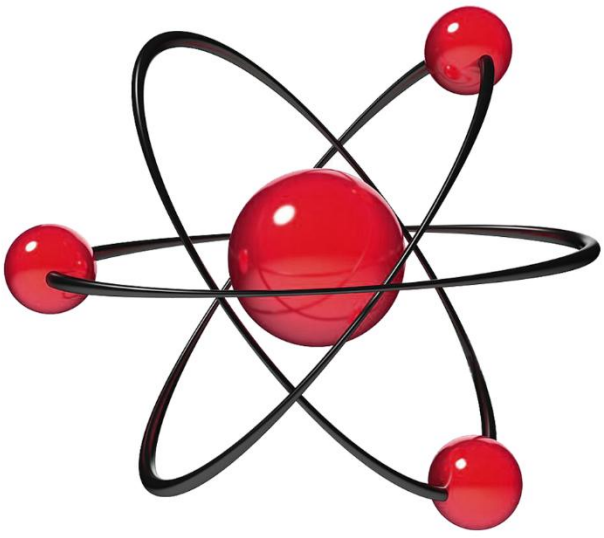
Adobe Animate 2024
Action Script 3.0

Михайло Каленик

Adobe Animate
у навчанні фізики



Михайло Каленик

Adobe Animate у навчанні фізики

НАВЧАЛЬНИЙ ПОСІБНИК
(електронне видання)

Київ
2026

УДК 373.5.091.2:53](477)(072.034.2)=111
К17

Рекомендовано до видання вченою радою Сумського державного педагогічного університету імені А.С. Макаренка (протокол № 8 від 2 березня 2026 р.)

Рецензенти:

Горошко Юрій Васильович – доктор педагогічних наук, професор, завідувач кафедри інформатики і обчислювальної техніки, Національний університет "Чернігівський колегіум" імені Т.Г. Шевченка
Однодворець Лариса Валентинівна – доктор фізико-математичних наук, професор, завідувачка кафедри електроніки, загальної та прикладної фізики Сумського державного університету.

К17 **Каленик М.В. Adobe Animate у навчанні фізики**, навчальний посібник [Електронне видання], – Київ, Видавець Цибульська, 2026. – 113 с.

ISBN 978-617-8324-74-2

Навчальний посібник «Adobe Animate у навчанні фізики» призначений для здобувачів фізико-математичних факультетів педагогічних університетів, майбутніх учителів фізики, а також викладачів, які прагнуть впроваджувати сучасні інформаційні технології у навчальний процес. У посібнику розкрито теоретичні та практичні основи використання Adobe Animate для візуалізації фізичних явищ, моделювання експериментів та створення інтерактивних симуляцій. Особлива увага приділяється розвитку ІКТ-компетентностей здобувачів, формуванню навичок програмування на ActionScript 3.0, організації проєктної діяльності та інтеграції анімацій у різні форми навчання фізики. Посібник містить покрокові інструкції, практичні завдання, методичні рекомендації щодо створення анімацій, а також приклади реалізації фізичних моделей різної складності. Матеріали сприяють підвищенню мотивації до вивчення фізики, розвитку критичного мислення, творчості й дослідницьких навичок. Посібник стане корисним ресурсом для викладачів, здобувачів і всіх, хто зацікавлений у сучасних підходах до фізичної освіти з використанням мультимедійних технологій.

УДК 373.5.091.2:53](477)(072.034.2)=111

ISBN 978-617-8324-74-2

Зміст

Вступ

У XXI столітті освіта перебуває у стані динамічних змін, що зумовлені стрімким розвитком інформаційних технологій, цифровізацією суспільства й оновленням вимог до професійної підготовки майбутніх фахівців. Фізика, як фундаментальна наука, відіграє ключову роль у формуванні наукового світогляду, розвитку критичного мислення та дослідницьких навичок. Проте традиційні підходи до викладання фізики часто не відповідають сучасним потребам здобувачів, які зростають у цифровому середовищі, мають високий рівень візуальної культури та очікують інтерактивності й наочності від навчального процесу.

Використання інформаційно-комунікаційних технологій (ІКТ) у навчанні фізики дозволяє зробити освітній процес більш ефективним, цікавим і доступним. Сучасні програмні засоби дають змогу візуалізувати складні фізичні явища, моделювати експерименти, створювати інтерактивні симуляції, що сприяє глибшому розумінню навчального матеріалу та формуванню стійкої мотивації до вивчення фізики. Особливої актуальності набуває підготовка майбутніх учителів фізики до використання таких технологій у своїй професійній діяльності.

Традиційно для ілюстрації фізичних явищ використовувалися малюнки на дошці, плакати, статичні слайди, а також демонстраційні експерименти. Однак ці засоби мають суттєві обмеження: вони не завжди дозволяють відтворити динаміку процесу, показати приховані або швидкоплинні явища, забезпечити індивідуалізацію навчання. З появою комп’ютерної графіки, анімації та мультимедійних технологій з’явилася можливість створювати динамічні моделі, які наочно демонструють перебіг фізичних процесів у реальному часі, дають змогу експериментувати з параметрами, досліджувати різні сценарії розвитку подій.

Візуалізація в освіті – це не лише засіб ілюстрації, а й потужний інструмент для розвитку уяви, просторового мислення, аналітичних здібностей, уміння працювати з інформацією. В умовах реформування освіти та впровадження компетентнісного підходу особливого значення набуває формування у здобувачів вміння самостійно створювати мультимедійний контент, використовувати його для пояснення складних понять, організації навчальних проєктів, проведення досліджень.

Серед сучасних програмних засобів для створення анімації й інтерактивних моделей особливе місце посідає Adobe Animate. Це універсальна платформа для розробки 2D-анімації, яка поєднує зручний інтерфейс, потужний інструментарій для роботи з графікою та можливості програмування на ActionScript 3.0. Adobe Animate дозволяє створювати як прості анімовані ролики, так і складні інтерактивні симуляції, що ідеально підходить для візуалізації фізичних явищ і процесів.

На відміну від багатьох інших програм, Adobe Animate забезпечує гнучкість у побудові анімацій, дає змогу інтегрувати різні типи медіа

(зображення, звук, відео), створювати навчальні ігри, віртуальні лабораторії, інтерактивні презентації. Завдяки підтримці ActionScript 3.0 можна програмувати поведінку об'єктів, задавати фізичні закони, керувати взаємодією користувача з моделлю. Це відкриває широкі можливості для творчої діяльності здобувачів, розвитку їхніх ІКТ-компетентностей, реалізації навчальних проєктів різної складності.

Сучасний учитель фізики має бути не лише носієм знань, а й організатором навчального середовища, здатним використовувати новітні технології для підвищення ефективності навчання, розвитку пізнавальної активності учнів, формування навичок самостійної та проєктної діяльності. Опанування Adobe Animate дає майбутньому вчителю низку переваг:

- можливість створювати власні анімації, що враховують специфіку навчального матеріалу, рівень підготовки учнів, дидактичні цілі;
- здатність розробляти інтерактивні моделі для проведення віртуальних експериментів, досліджень, демонстрацій;
- уміння організовувати навчальні проєкти з використанням мультимедійних технологій, залучати учнів до створення власних анімацій та симуляцій;
- підвищення мотивації до навчання фізики через використання сучасних, цікавих і доступних засобів візуалізації;
- розвиток професійної мобільності, здатності швидко адаптуватися до вимог сучасної школи та освітнього середовища.

Метою цього посібника є формування у майбутніх учителів фізики практичних навичок створення анімацій та інтерактивних моделей фізичних явищ за допомогою Adobe Animate, а також ознайомлення з основами програмування на ActionScript 3.0 для розробки навчальних симуляцій. Посібник орієнтований на здобувачів спеціальності "Середня освіта (Фізика)", які прагнуть оволодіти сучасними засобами візуалізації та інтерактивного моделювання у професійній діяльності.

Завдання посібника:

- Ознайомити з можливостями Adobe Animate у навчанні фізики;
- Навчити основам роботи з інтерфейсом та інструментами програми;
- Дати базові уявлення про ActionScript 3.0 та його застосування для створення інтерактивних моделей;
- Показати переваги Adobe Animate у порівнянні з іншими програмами для анімації фізичних процесів;
- Розвинути вміння створювати власні анімації та інтерактивні моделі на конкретних прикладах фізичних явищ;
- Сприяти формуванню проєктних, дослідницьких, ІКТ-компетентностей здобувачів.

Структура посібника передбачає поєднання теоретичних і практичних розділів, поступове ускладнення навчальних завдань, інтеграцію лекційних і практичних занять, виконання індивідуальних та групових проєктів різної тривалості (міні-, короткострокових, тижневих, довгострокових).

Очікувані результати навчання

Після опрацювання матеріалів посібника ви зможете:

- Впевнено працювати з Adobe Animate для створення анімацій та інтерактивних моделей;
- Використовувати ActionScript 3.0 для програмування навчальних симуляцій;
- Розробляти власні навчальні проєкти з анімації фізичних процесів різної складності;
- Обґрунтовувати вибір програмних засобів для візуалізації фізичних явищ у навчанні;
- Організовувати навчальні проєкти з використанням мультимедійних технологій;
- Використовувати анімацію як ефективний інструмент у професійній діяльності вчителя фізики.

Переваги Adobe Animate у порівнянні з іншими програмними засобами

Adobe Animate вигідно відрізняється від багатьох альтернативних програм завдяки:

- гнучкості у створенні як простих, так і складних анімацій;
- можливості програмування поведінки об’єктів і створення інтерактивності;
- підтримці різних форматів експорту (HTML5, відео, GIF, Flash тощо);
- інтеграції з іншими продуктами Adobe та сторонніми ресурсами;
- широкому спектру навчальних матеріалів, спільнот, готових шаблонів.

Ці переваги роблять Adobe Animate універсальним інструментом для візуалізації фізичних явищ, організації навчальних проєктів, розвитку творчого потенціалу як здобувачів, так і майбутніх учителів фізики.

У цьому посібнику ви знайдете покрокові інструкції, практичні завдання, реальні приклади з анімації фізичних процесів, методичні рекомендації щодо інтеграції анімації у навчальний процес, а також поради щодо організації проєктної діяльності різної тривалості та складності. Сподіваємося, що опанування матеріалів допоможе вам стати сучасним, творчим і конкурентоспроможним фахівцем у галузі фізичної освіти.

Теоретичні основи використання інформаційних технологій у навчанні фізики

У XXI столітті освіта переживає глибокі трансформації, пов'язані з цифровізацією суспільства, розвитком інформаційних технологій та зростанням вимог до якості й гнучкості навчального процесу. Фізика як фундаментальна наука потребує не лише передавання знань, а й формування наукового світогляду, розвитку критичного мислення, дослідницьких і проєктних компетентностей. Однак традиційні методи викладання часто не здатні забезпечити належний рівень мотивації, наочності та індивідуалізації навчання.

Інформаційно-комунікаційні технології (ІКТ) – це сукупність апаратних і програмних засобів, які забезпечують збір, зберігання, обробку, передачу, поширення й використання інформації. У контексті освіти ІКТ охоплюють комп'ютери, мультимедійні засоби, інтернет, мобільні пристрої, програмне забезпечення для моделювання, візуалізації, тестування та контролю знань.

Використання ІКТ у навчанні фізики дозволяє:

- Підвищити мотивацію та інтерес до предмета завдяки інтерактивності, візуалізації, ігровим елементам.
- Розширити можливості для експерименту: моделювати явища, які складно чи неможливо відтворити в реальних умовах (наприклад, атомні процеси, космічні явища).
- Індивідуалізувати навчання: адаптувати темп, складність, спосіб подання матеріалу до потреб учня.
- Підвищити об'єктивність контролю знань через автоматизовані тести, симуляції, електронні журнали.
- Забезпечити доступ до сучасних наукових даних та світових освітніх ресурсів через інтернет.

Дослідження свідчать, що впровадження ІКТ у навчальний процес з фізики підвищує ефективність практичних і лабораторних занять приблизно на 20%, а об'єктивність контролю знань – на 15-20%. Комп'ютерна візуалізація матеріалу сприяє формуванню уявлень, що лежать в основі образного мислення, полегшує засвоєння складних понять, розвиває творчу активність здобувачів. Важливо, що ІКТ не лише підсилюють бажання до навчання, а й роблять предмет ближчим, сучаснішим, емоційно привабливим.

Основні напрями використання ІКТ у фізиці:

- Супровід лекцій і демонстрацій: використання анімацій, відеофрагментів, інтерактивних презентацій для пояснення складних явищ.
- Комп'ютерне моделювання: створення моделей фізичних процесів, які можна досліджувати, змінювати параметри, аналізувати результати.
- Віртуальні лабораторії: проведення експериментів у цифровому середовищі, що розширює межі досліджень.
- Самостійна робота здобувачів: виконання завдань, тестів, проєктів із

використанням освітніх платформ і програмного забезпечення.

- Телекомунікаційні технології: дистанційне навчання, участь у вебінарах, онлайн-олімпіадах, спільна робота над проєктами.

Візуалізація – це процес створення наочних образів, які відображають суть фізичних явищ і процесів. У фізиці вона дозволяє:

- Перетворити абстрактні поняття на зрозумілі моделі.
- Демонструвати динаміку явищ у реальному часі.
- Пояснювати взаємозв'язки між величинами, параметрами, умовами.
- Забезпечити доступність матеріалу для різних типів сприйняття.

Програмні засоби для візуалізації фізики можна умовно поділити на такі групи:

- Програми для створення 2D- та 3D-анімації (Adobe Animate, Blender, Toon Boom Harmony, Moho).
- Системи комп'ютерного моделювання (Algodoo, PhET Interactive Simulations, Crocodile Physics).
- Динамічні математичні середовища (GeoGebra, Mathcad).
- Презентаційні платформи (PowerPoint, Google Slides).
- Програмні середовища для програмування та інтерактивних проєктів (Scratch, Unity, Processing).
- AR/VR-технології (доповнена і віртуальна реальність).

Порівняємо основні програмні продукти.

Adobe Animate

- Тип: 2D-анімація, інтерактивність, програмування (ActionScript).
- Можливості: покадрова анімація, tween-анімація, створення інтерактивних моделей, робота з шарами, символами, скелетна анімація, програмування поведінки об'єктів.
- Переваги: гнучкість, інтеграція з іншими продуктами Adobe, підтримка різних форматів експорту (HTML5, відео, GIF, SWF), можливість створювати навчальні ігри та симуляції.

Blender

- Тип: 3D-моделювання, анімація, рендеринг.
- Можливості: створення статичних і динамічних 3D-моделей, анімація руху тіл, візуалізація складних механічних, оптичних, електромагнітних процесів.
- Переваги: безкоштовність, потужний інструментарій для 3D-візуалізації, підтримка фізичних рушіїв (симуляція твердих тіл, рідин, газів).
- Обмеження: складний інтерфейс, потребує високої комп'ютерної грамотності, не призначений для швидкого створення простих навчальних анімацій.

Toon Boom Harmony, Moho

- Тип: професійна 2D-анімація.
- Можливості: покадрова анімація, ріггінг, скелетна анімація, робота з шарами, камерами, ефектами.
- Переваги: високий рівень деталізації, придатність для створення мультфільмів, науково-популярних відео.

- Обмеження: складність освоєння, орієнтація на професіоналів, висока вартість.

Algodoo

- Тип: фізичне моделювання, 2D-середовище.
- Можливості: створення інтерактивних моделей механічних систем, симуляція руху тіл, зіткнень, пружин, рідин.
- Переваги: простота інтерфейсу, швидке моделювання, інтерактивність, орієнтація на освіту.
- Обмеження: обмежена візуальна гнучкість, відсутність складної анімації, немає програмування поведінки об'єктів.

PhET Interactive Simulations

- Тип: онлайн-симуляції фізичних і хімічних процесів.
- Можливості: готові інтерактивні моделі для різних розділів фізики, можливість змінювати параметри, спостерігати результати.
- Переваги: безкоштовність, простота використання, висока якість наукової візуалізації.
- Обмеження: неможливість створювати власні моделі, обмеження готовим набором симуляцій.

GeoGebra

- Тип: динамічна математика, геометрія, алгебра.
- Можливості: побудова графіків, моделювання руху, робота з параметрами.
- Переваги: безкоштовність, простота, інтеграція з математикою.
- Обмеження: обмежена анімація, відсутність програмування складної поведінки.

PowerPoint, Google Slides

- Тип: презентаційні засоби.
- Можливості: створення слайдів, простих анімацій, вставка відео, зображень.
- Переваги: доступність, простота, знайомий інтерфейс.
- Обмеження: дуже обмежені можливості для складної анімації, відсутність інтерактивності, немає програмування.

Scratch

- Тип: візуальне програмування, інтерактивні проєкти.
- Можливості: створення простих ігор, моделей, анімацій за допомогою блоків.
- Переваги: простота, орієнтація на початківців, навчання основам алгоритмізації.
- Обмеження: обмежений набір інструментів для фізики, низька деталізація анімацій.

AR/VR-технології

- Тип: доповнена та віртуальна реальність.
- Можливості: створення інтерактивних 3D-сцен, занурення у віртуальні лабораторії, моделювання складних явищ.
- Переваги: максимальна наочність, ефект присутності, розвиток

просторового мислення.

- Обмеження: висока вартість обладнання, складність розробки, потреба у спеціальних навичках.

Adobe Animate поєднує можливості покадрової анімації, tween-анімації, скелетної анімації, роботи з шарами, символами, а також програмування поведінки об'єктів за допомогою ActionScript 3.0. Це дозволяє створювати як прості ілюстрації, так і складні інтерактивні моделі фізичних явищ, включаючи навчальні ігри, лабораторні роботи, симуляції експериментів.

На відміну від більшості освітніх програм, Adobe Animate підтримує повноцінне ***програмування (ActionScript 3.0)***, що дозволяє:

- Реалізовувати складну логіку поведінки об'єктів.
- Створювати інтерактивні елементи: кнопки, слайдери, поля введення, інтерактивні графіки.
- Відтворювати фізичні закони у динаміці, моделювати експерименти з можливістю зміни параметрів у реальному часі.

Adobe Animate забезпечує:

- Професійний рівень графіки та анімації.
- Підтримку векторної графіки, що дозволяє масштабувати зображення без втрати якості.
- Додавання спецефектів, фільтрів, прозорості, анімації камери.

Інтеграція та експорт:

- Можливість експортувати проєкти у різних форматах: HTML5, відео, GIF, SWF, інтерактивні веб-сторінки.
- Інтеграція з іншими продуктами Adobe (Photoshop, Illustrator, After Effects), що дозволяє використовувати готові графічні ресурси.

Практичні переваги для навчання фізики:

- Створення динамічних моделей: наприклад, моделювання руху тіл, коливань, хвиль, електричних і магнітних полів.
- Візуалізація абстрактних понять: наприклад, процесів у мікросвіті, динаміки змін, взаємодії об'єктів.
- Реалізація віртуальних лабораторій: можливість створювати симуляції експериментів із керуванням параметрами, збором і аналізом даних.
- Розвиток ІКТ-компетентностей у здобувачів: навчання основам анімації, програмування, проєктної діяльності.

Інформаційно-комунікаційні технології стали невід'ємною складовою сучасної фізичної освіти, сприяючи підвищенню мотивації, якості засвоєння знань, розвитку дослідницьких і творчих компетентностей. Візуалізація фізичних явищ за допомогою сучасних програмних засобів відкриває нові горизонти для навчання: від простих ілюстрацій до складних інтерактивних моделей і симуляцій. Серед усього розмаїття програмних продуктів Adobe Animate вирізняється гнучкістю, можливістю програмування, високою якістю графіки й універсальністю, що робить його потужним інструментом для підготовки майбутніх учителів фізики та організації сучасного навчального процесу. Опанування цього інструменту дозволяє не лише осучаснити уроки фізики, а й надає навички, затребувані в цифровому світі.

Сучасні інформаційні технології, зокрема Adobe Animate, можуть зробити викладання фізики не лише ефективнішим, а й справді захопливим.

Фізика – це наука про рух, сили, енергію, явища, які часто відбуваються дуже швидко, або навпаки – надто повільно, щоб ми могли їх спостерігати без допомоги техніки. Уявіть, як складно пояснити учням, що таке хвиля, електромагнітне поле чи рух тіл під дією складних сил, якщо обмежитися лише словами чи статичними малюнками.

Тут на допомогу приходить анімація – динамічний, живий спосіб показати, як змінюються фізичні величини, як розвиваються процеси у часі. Вона допомагає не просто запам'ятати формули, а побачити фізику в дії.

В Adobe Animate можна використовувати ***чотири основні типи анімації:***

- Покадрова анімація, коли кожен кадр намальований окремо – це як створення мультфільму.
- Tween-анімація, де комп'ютер автоматично створює проміжні кадри між ключовими станами.
- Скелетна анімація, що дозволяє анімувати об'єкти як систему пов'язаних частин (наприклад, рух руки).
- Інтерактивні моделі, коли учень може змінювати параметри і бачити, як це впливає на процес.

З педагогічної точки зору використання анімацій у навчанні фізики дозволяє:

- Покращити розуміння складних процесів. Коли учень бачить, як змінюється швидкість тіла чи як коливається маятник, у нього формується глибше уявлення, ніж від сухого тексту.
- Розвивати просторове мислення. Анімації допомагають уявити рух у тривимірному просторі, взаємодію об'єктів.
- Підвищити мотивацію. Динамічний візуальний матеріал цікавіший і спонукає до активної участі.
- Сприяти формуванню дослідницьких навичок. Створення або аналіз анімацій – це вже маленький науковий проєкт.

Уявіть, що ви пояснюєте закон збереження імпульсу. Замість того, щоб малювати на дошці складні схеми, ви показуєте анімацію зіткнення двох тіл, де учні бачать, як змінюються швидкості. Це не просто ілюстрація – це спосіб зробити лекцію живою і зрозумілою.

На практичних заняттях можна використовувати готові анімації для аналізу, а також навчати здобувачів освіти створювати власні. Наприклад, розбити клас на групи, кожна з яких створює анімацію певного фізичного явища – це і практика, і творчість, і співпраця.

Учні можуть отримувати домашні завдання зі створення анімацій, що розвиває їхні ІКТ-компетентності та вміння застосовувати знання на практиці. Це також формує відповідальність і самостійність.

Викладач може інтегрувати анімації у різні форми навчання:

- Фронтальне навчання: демонстрація анімацій викладачем.
- Групова робота: спільне створення анімацій.

- Індивідуальне навчання: самостійне опанування інструментів.
- Дистанційне навчання: використання анімацій у відеоуроках і онлайн-курсах.

Навчання на основі проєктів – це не просто модний тренд, а перевірений спосіб залучити учнів у глибоке вивчення предмета. Проєкти допомагають поєднати теорію з практикою, розвивають критичне мислення, навички командної роботи.

Як організувати проєкт із Adobe Animate?

1. Вибір теми. Наприклад, моделювання руху тіла, електромагнітних хвиль або коливань.
2. Планування. Визначення цілей, розподіл ролей у групі.
3. Ознайомлення з Adobe Animate. Навчання базовим інструментам і програмуванню.
4. Створення анімації. Розробка моделі, додавання інтерактивності.
5. Тестування і вдосконалення. Перевірка роботи, внесення змін.
6. Презентація. Демонстрація результатів, обговорення.
7. Рефлексія. Аналіз досвіду, обговорення труднощів і успіхів.

Коли учні створюють анімації, оцінювати треба не лише технічну якість, а й глибину розуміння фізики, творчий підхід, здатність пояснювати свої рішення. При цьому ***критеріями оцінювання*** можуть бути:

- Коректність фізичної моделі. Чи правильно відображено явище?
- Технічна якість анімації. Чи плавно і зрозуміло рухаються об’єкти?
- Інтерактивність. Чи є можливість змінювати параметри і бачити результат?
- Оригінальність і творчість. Чи є щось особливе, що відрізняє роботу від інших?
- Презентація. Чи вміє учень чітко і логічно пояснити свою роботу?

Чому Adobe Animate – це must-have у вашому арсеналі?

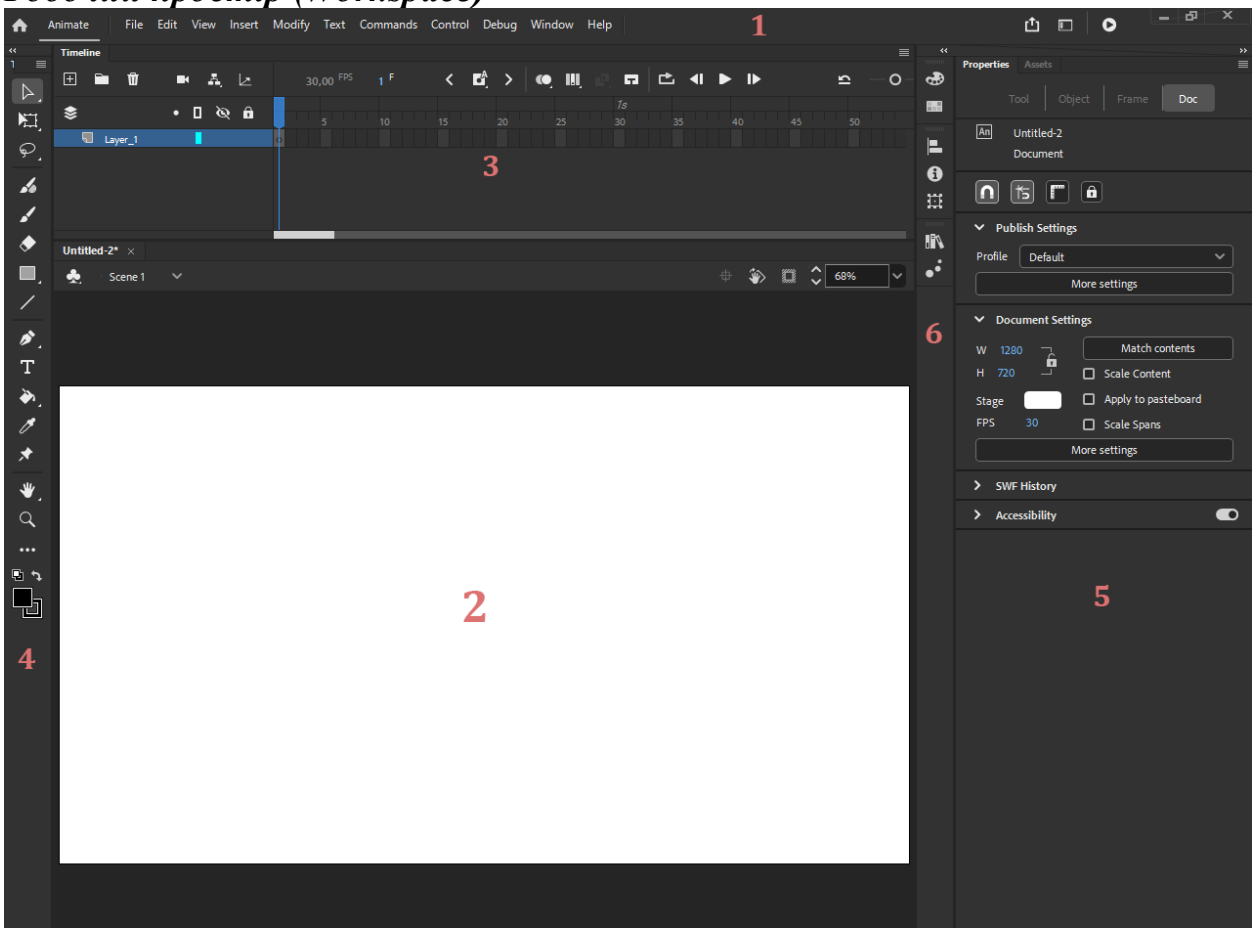
- Це інструмент, який поєднує простоту і потужність: від створення базових анімацій до складних інтерактивних моделей.
- Він допомагає формувати не лише знання фізики, а й ІКТ-компетентності, навички програмування, проєктної діяльності.
- Використання анімацій робить уроки більш наочними і цікавими для учнів, що підвищує їхню мотивацію і якість засвоєння матеріалу.
- Adobe Animate відкриває широкі можливості для творчості і самовираження як учнів, так і вчителя.
- Володіння цим інструментом допоможе вам бути сучасними педагогами, готовими до викликів цифрової освіти.

Основи роботи з Adobe Animate

Adobe Animate – це сучасний програмний продукт для створення 2D-анімації, інтерактивних моделей та мультимедійних проєктів. Його потужний інструментарій дозволяє не лише малювати й анімувати об'єкти, а й програмувати їхню поведінку, створювати симуляції фізичних процесів, що є надзвичайно цінним для викладання фізики. Опанування Adobe Animate відкриває перед учителем можливості для візуалізації абстрактних понять, моделювання експериментів, створення інтерактивних навчальних середовищ.

Загальна структура інтерфейсу Adobe Animate

Робочий простір (Workspace)



Робочий простір Adobe Animate складається з низки панелей, вікон та інструментів, які можна налаштовувати під власні потреби. Будь-яке розташування елементів інтерфейсу називається *робочим простором* (workspace). Для анімації найзручніше використовувати шаблон «Classic», який можна обрати через меню Windows => Workspaces => Classic.

Основні компоненти інтерфейсу

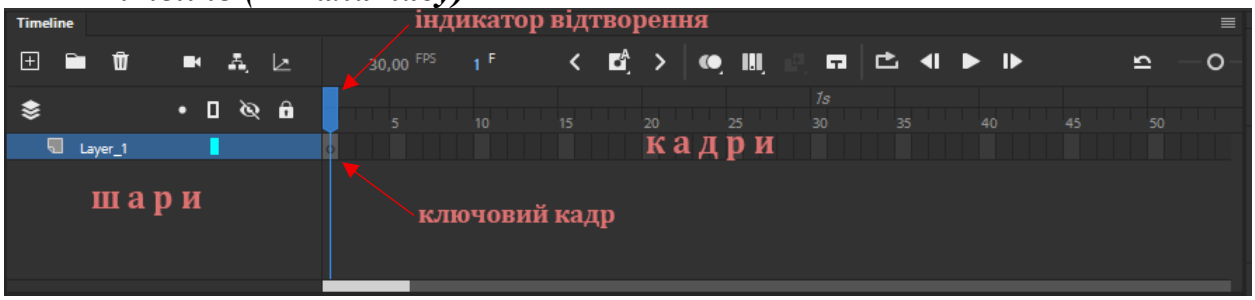
- ***Application Bar*** – верхня панель із меню, перемикачем робочих просторів та системними командами (**1**).
- ***Stage (Сцена)*** – центральна область, де розміщуються й анімуються об'єкти. Це «полотно», на якому створюється анімація (**2**).

- ***Timeline (Шкала часу)*** – панель, що дозволяє керувати часом, шарами та ключовими кадрами анімації (**3**).
- ***Tools Panel (Панель інструментів)*** – вертикальна панель ліворуч із основними інструментами для малювання, вибору, трансформації, тощо (**4**).
- ***Properties Panel (Панель властивостей)*** – відображає властивості вибраного об'єкта, дає змогу змінювати параметри (колір, розмір, положення тощо) (**5**).
- ***Library Panel (Бібліотека)*** – містить усі створені символи, імпортовані зображення, звуки та інші ресурси проекту (**6**).

Stage (Сцена)

Сцена – це головне робоче поле, на якому розміщуються всі об'єкти анімації. Її розміри можна змінювати в налаштуваннях документа. На сцені можна розташовувати графічні об'єкти, текст, символи, імпортовані зображення, відео та інші елементи.

Timeline (Шкала часу)



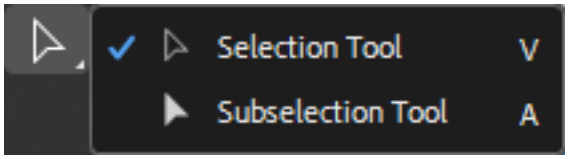
Шкала часу складається з:

- ***Шарів (Layers)*** – дозволяють розміщувати різні об'єкти на окремих «плівках», що полегшує анімацію та організацію проекту.
- ***Кадрів (Frames)*** – одиниці часу, у яких розміщуються об'єкти. Один кадр – це одна «миттєвість» анімації.
- ***Ключових кадрів (Keyframes)*** – спеціальні кадри, у яких відбуваються зміни (початок/кінець руху, зміна властивостей тощо).
- ***Playhead (Індикатор відтворення)*** – показує поточний кадр анімації.

Можна додавати  , видаляти  , переміщати кадри (ліва кнопка миші), створювати анімацію за допомогою покадрової або автоматичної (tweened) анімації (клікнути правою кнопкою миші по ключовому кадру).

Панель інструментів (Tools Panel): огляд та призначення

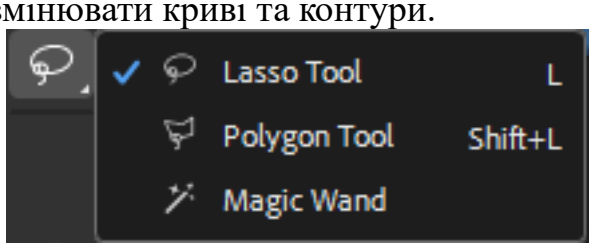
Інструменти вибору



- ***Selection Tool (V)*** – основний інструмент для виділення й

переміщення об'єктів на сцені. Дозволяє виділяти цілі об'єкти або їх частини, змінювати розмір, повертати.

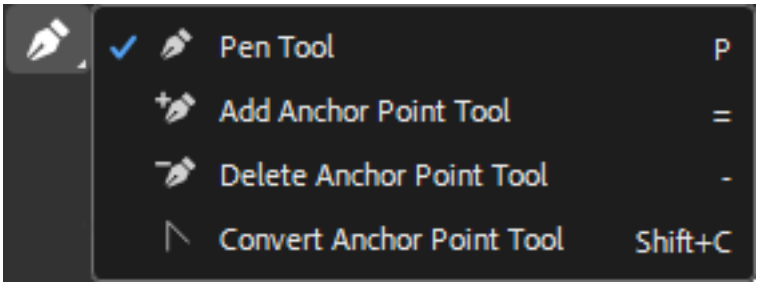
- ***Subselection Tool (A)*** – дозволяє редагувати окремі вузли та сегменти векторних об'єктів, змінювати криві та контури.



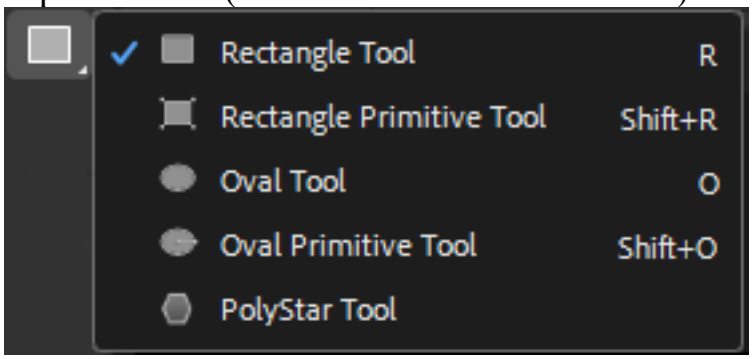
- ***Lasso Tool (L)*** – для довільного виділення частин об'єктів за допомогою малювання замкненої області.
- ***Polygon Tool (Shift+L)*** – виділення за допомогою багатокутника для точного вибору складних областей.
- ***Magic Wand Tool*** – виділення областей за кольором, корисно для роботи з однотонними ділянками.

Інструменти малювання

-  ***Brush Tool (B)*** – малювання вільних ліній із можливістю налаштування форми, розміру, кута кисті. Дозволяє створювати природні мазки.
-  ***Line Tool (N)*** – малювання вільних ліній, схоже на олівець. Лінії можна редагувати після нанесення.

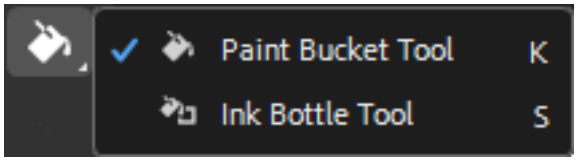


- ***Pen Tool (P)*** – малювання прямих і кривих ліній із точним контролем за допомогою опорних точок (аналогічно Adobe Illustrator).

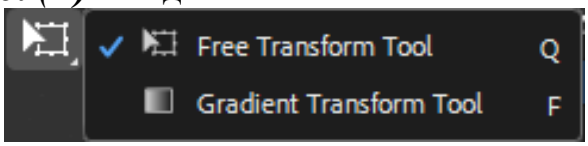


- ***Rectangle Tool (M)*** – створення прямокутників і квадратів.
- ***Oval Tool (O)*** – створення кіл і еліпсів.

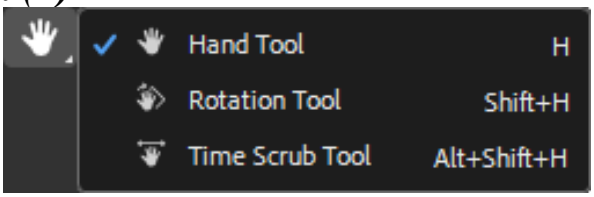
Інші інструменти



- ***Paint Bucket Tool (K)*** – заливка замкнених областей кольором.
-  ***Eyedropper Tool (I)*** – вибір кольору з будь-якого об'єкта на сцені.
-  ***Eraser Tool (E)*** – видалення частин малюнка.



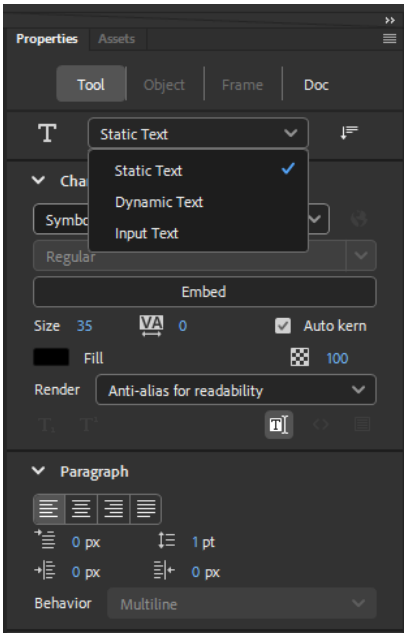
- ***Free Transform Tool (Q)*** – масштабування, обертання, нахил об'єктів.
-  ***Zoom Tool (Z)*** – збільшення/зменшення масштабу сцени.



- ***Hand Tool (H)*** – переміщення сцени у вікні перегляду.

Спеціальні інструменти

-  ***Text Tool (T)*** – дозволяє створювати та редагувати текстові поля на сцені. Можна створювати як однорядковий текст (point text), так і багаторядковий (paragraph text), задаючи ширину текстового блоку.



На панелі властивостей (properties) можна створювати різні типи тексту:

Static Text – статичний текст, який не змінюється під час виконання анімації.

Dynamic Text – текст, який можна змінювати програмно (наприклад, виводити змінні, результати обчислень).

Input Text – текстове поле для введення користувачем (корисно для інтерактивних моделей).

Також можна використати додткові можливості:

Редагування тексту: підтримуються стандартні операції копіювання, вставки, вирівнювання, зміни шрифту, розміру, кольору, стилю (жирний, курсив), міжрядкових інтервалів, відступів.

Властивості тексту: можна обрати шрифт, розмір, колір, тип вирівнювання, а також напрямок тексту (горизонтальний або вертикальний).

Фільтри та ефекти: до тексту можна застосовувати тіні, розмиття, світіння, рельєфи, що робить його більш виразним і привабливим для глядача.

Розбиття тексту на символи: для створення анімації окремих букв

можна розбити текст на символи (*Break Apart – шляхом натискання клавіш Ctrl + B, повторне натискання перетворює текст на графіку*). Це дозволяє анімувати кожну букву окремо, змінювати її форму, колір, положення.

Анімація тексту: використовуючи ключові кадри, tween-анімацію, а також програмування на ActionScript, можна створювати ефекти появи, зникнення, руху, трансформації тексту.

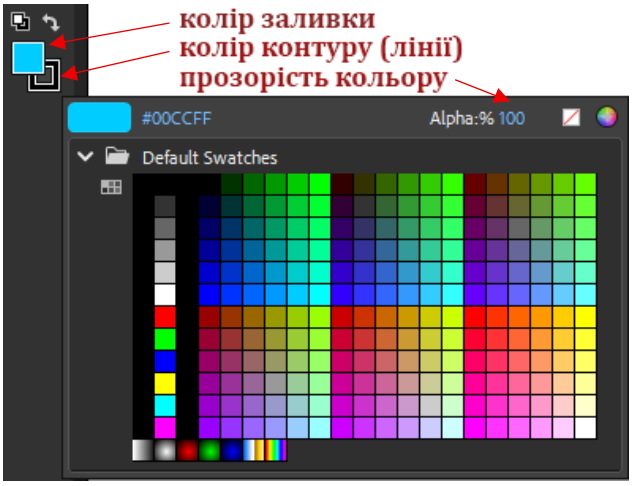
-  ***Asset Warp Tool (W)*** – інструмент для деформації векторних об'єктів або символів Adobe Animate за допомогою сітки контрольних точок (warp mesh). Він дозволяє гнучко змінювати форму об'єктів, створювати плавні вигини, скручування, розтягнення, що особливо корисно для анімації складних рухів і деформацій.

  Основні можливості Asset Warp Tool:

  - *Створення сітки деформації:* при виборі об'єкта інструмент автоматично накладає на нього сітку з контрольними точками.
  - *Редагування сітки:* можна додавати, видаляти точки, переміщувати їх для локальної деформації об'єкта.
  - *Анімація деформації*: за допомогою ключових кадрів можна анімувати зміну положення контрольних точок, створюючи плавні переходи між формами.
  - *Підтримка векторної графіки та символів:* деформації застосовуються до векторних об'єктів, а також до символів, що дозволяє створювати складні анімації без потреби малювати кожен кадр вручну.

  Робота з сіткою проста і візуально зрозуміла, що дозволяє швидко освоїти інструмент.

-  ***Color*** – інструменти для вибору/зміни кольору заливки (***Fill Color***) та кольору контуру або ліній (***Stroke Color***). При виборі інструменту розгорнеться віконце вибору кольорів, палітр, прозорості.

- 

Панель властивостей (Properties Panel)

Панель властивостей – універсальний інструмент для налаштування параметрів вибраного об'єкта або інструмента. В залежності від того, що вибрано (об'єкт, шар, інструмент), панель змінює свій вигляд і дозволяє змінювати:

- колір заливки та обведення
- товщину лінії
- положення, розмір, обертання об'єкта
- властивості анімації (тривалість, тип tween-інтерполяції)
- параметри тексту (шрифт, розмір, вирівнювання)
- властивості символів та екземплярів

Бібліотека (Library Panel)

Бібліотека містить усі ресурси проекту: символи, графічні об'єкти, звуки, відео, імпортовані файли. З бібліотеки можна перетягувати об'єкти на сцену, створювати копії, організовувати ресурси за папками. Символи – це особливі об'єкти (графічні, кнопки, кліпи), які можна використовувати багаторазово, а зміни в символі автоматично відображаються у всіх його екземплярах на сцені.

Робота з шарами та кадрами

Організація шарів

Шари дозволяють розділяти різні елементи анімації (фон, об'єкти, текст, ефекти) для зручності редагування та анімації. Можна:

- створювати нові шари
- перейменовувати, видаляти, групувати
- блокувати/розблоковувати шари
- приховувати/відображати шари
- змінювати порядок шарів (перетягуванням)

Робота з ключовими кадрами

Ключові кадри визначають моменти змін у анімації. Між ключовими кадрами програма автоматично створює проміжні стани (tweening). Можна створювати:

- ***Blank Keyframe –*** порожній ключовий кадр
- ***Motion Tween –*** анімація руху об'єкта між двома ключовими кадрами
- ***Shape Tween –*** анімація зміни форми об'єкта

При натисканні правою кнопкою миші на ключовий кадр з’явиться ***контекстне меню***, яке дозволяє:

Create Classic Tween – створити просту класичну анімацію;
Create Motion Tween – створити анімацію руху між ключовими кадрами;
Create Shape Tween – створити анімацію форми між ключовими кадрами;
Insert Frame (F5) – вставити кадр;
Remove Frames (Shift+F5) – видалити кадр;
Insert Keyframe – вставити ключовий кадр;
Insert Blank Keyframe – вставити пустий ключовий кадр;

Clear Keyframe (Shift+F6) – очистити ключовий кадр;
Convert to Keyframes (F6) – перетворити кадр на ключовий;
Convert to Blank Keyframes (F7) – перетворити кадр на пустий ключовий;
Cut Frames (Ctrl+Alt+x) – вирізати кадр (кадри);
Copy Frames (Ctrl+Alt+c) – копіювати кадр (кадри);
Paste Frames (Ctrl+Alt+v) – вставити кадр (кадри);
Clear Frames (Alt+Backspase) – очистити кадр (кадри);
Select All Frames (Ctrl+Alt+a) – виділити усі кадри;
Actions (F9) – виклик панелі Action Script для написання коду для кадра.

Основні етапи створення анімації в Adobe Animate

1. ***Планування проекту*** **–** визначення мети, сценарію, необхідних об'єктів.
2. ***Створення нового документа*** **–** вибір типу (HTML5 Canvas, ActionScript 3.0), розміру, частоти кадрів, кольору фону.
3. ***Додавання й розміщення об'єктів на сцені*** **–** малювання, імпорт, використання бібліотеки.
4. ***Організація шарів та кадрів*** **–** створення структури анімації.
5. ***Застосування анімації*** **–** покадрова, tween-анімація, скелетна анімація.
6. ***Додавання інтерактивності*** **–** програмування поведінки об'єктів за допомогою ActionScript (для AS3.0) або JavaScript (для HTML5 Canvas).
7. ***Тестування та публікація проекту*** **–** перевірка роботи анімації, експорт у потрібному форматі (відео, GIF, HTML5, SWF).

Практичні поради щодо організації робочого простору

- Використовуйте шаблони робочих просторів залежно від типу завдань (анімування, малювання, програмування).
- Налаштовуйте розташування панелей під власний стиль роботи.
- Зберігайте власні робочі простори для швидкого перемикання між різними задачами.
- Використовуйте гарячі клавіші для швидкої роботи з інструментами.

Опанування інтерфейсу та основних інструментів Adobe Animate є фундаментом для подальшого створення навчальних анімацій, моделей і симуляцій фізичних явищ. Вміння ефективно використовувати робочий простір, панелі, інструменти малювання, вибору, трансформації та анімації дозволяє майбутньому вчителю фізики реалізовувати найрізноманітніші дидактичні завдання, підвищувати якість та наочність навчального процесу.

Практична робота 1. Створення простої анімації руху

Завдання: створити анімацію кочення кульки між двома стінами.

Загальна логіка створення простої анімації:

- ✓ ***створити об'єкти, які будуть приймати участь в анімації, кожен на окремому шарі (перший кадр кожного шару – початкові положення об'єктів в анімації);***
- ✓ ***перетворити об'єкти на символи (обов'язково ті об'єкти, які планується рухати);***
- ✓ ***встановити необхідні властивості для об'єктів (назви, колір, розміри, точки прив'язки тощо);***
- ✓ ***продублювати ключові кадри і встановити в них кінцеві положення об'єктів;***
- ✓ ***в першому кадрі шару з об'єктом (який буде рухатись) встановити тип анімації;***
- ✓ ***на панелі властивостей деталізувати анімацію (указати потрібні властивості анімації).***

Додаткове завдання: зробити передню стінку прозорою, щоб було видно як котиться кулька між стінами.

1. Готуємо робочий простір.

Запускаємо програму ***Adobe Animate***.

У стартовому вікні вибираємо параметр ***Full HD***.

У верхньому правому куті сцени вибираємо значення масштабу ***Fit in Window***.

На даний момент існує один шар ***Layer_1***, ***індикатор відтворення*** знаходиться на ***першому пустому ключовому кадрі*** даного шару.

2. Створюємо об'єкти на сцені.

- **Створюємо перший об'єкт – стіну:**

Для зручності дамо назву шару у відповідності до створюваного об'єкта. Двічі клікнемо лівою кнопкою миші по назві шару ***Layer_1,*** пишемо Stina_1 і натискаємо на клавіатурі клавішу ***Enter***.

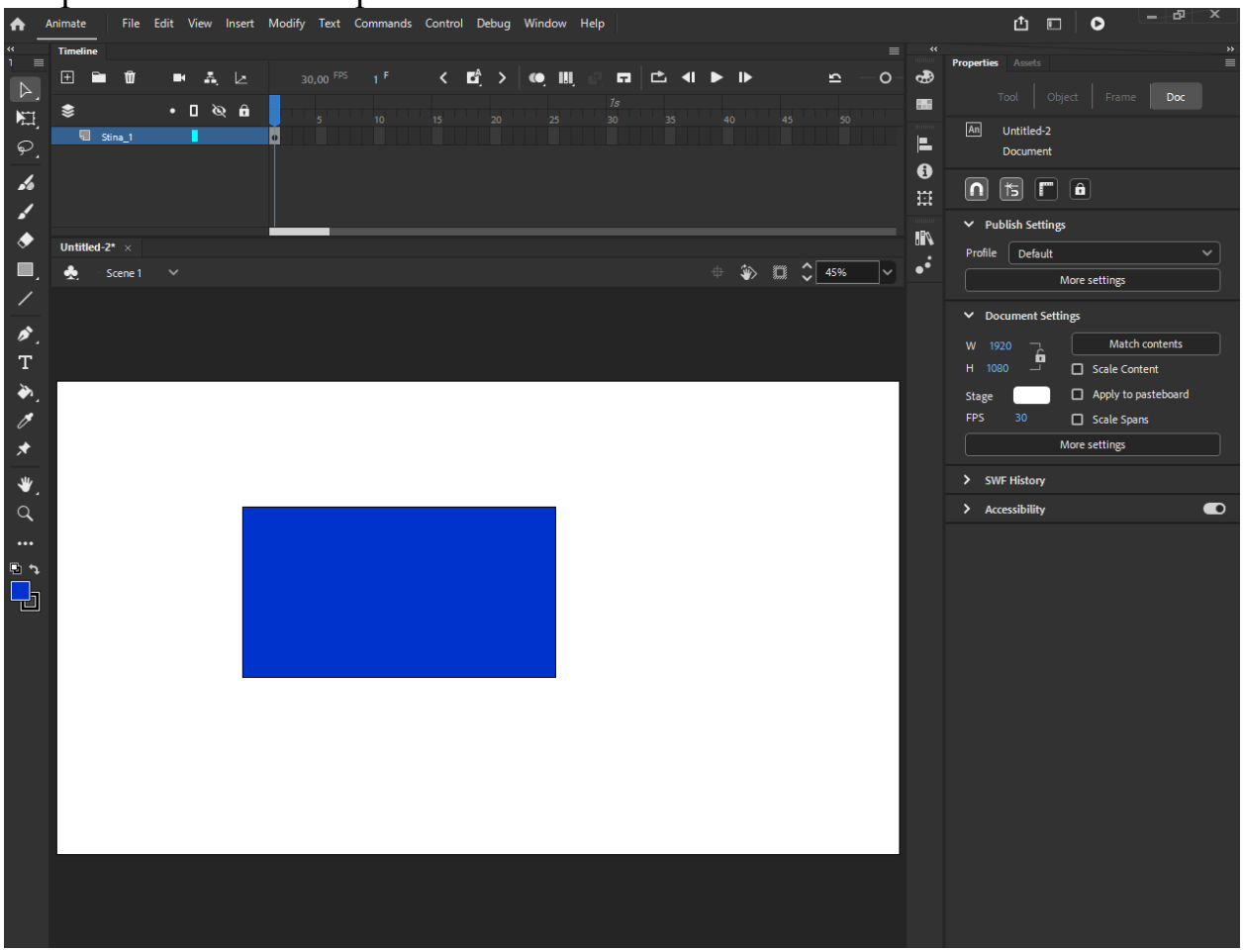
Відразу починаємо звикати до того, що усі назви та команди будемо писати латиницею.

За допомогою інструменту  ***Rectangle Tool*** малюємо прямокутник затиснувши ліву кнопку миші.

Вибираємо інструмент  ***Selection Tool*** і двічі клікаємо мишею в середині прямокутника – виділяємо його. Перш ніж змінювати або переміщувати об'єкт на сцені, його треба виділити.

Змінюємо колір прямокутника, натиснувши інструмент  ***Fill Color*** та

вибравши певний колір.



- **Створюємо другу стіну:**

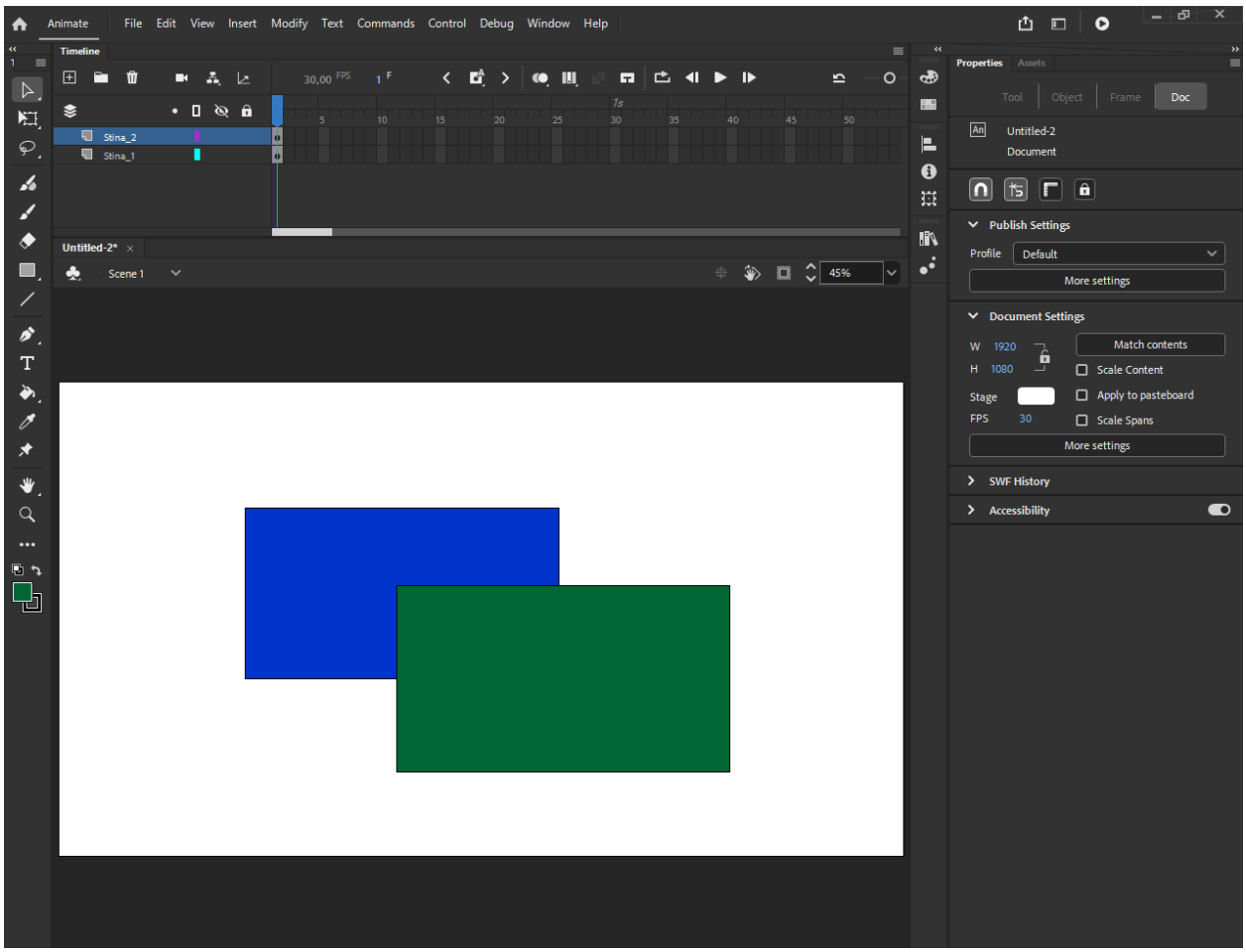
Додаємо ще один шар за допомогою інструмента  (знаходиться у верхній лівій частині ***Timeline***.

Змінюємо назву шару ***Layer_2*** на Stina_2.

За допомогою інструменту  ***Rectangle Tool*** малюємо прямокутник затиснувши ліву кнопку миші. Малювати починаємо приблизно із середини першого прямокутника.

Вибираємо інструмент  ***Selection Tool*** і двічі клікаємо мишею в середині прямокутника – виділяємо його. Якщо потрібно, можна відкоригувати розташування прямокутника (за допомогою миші або стрілок на клавіатурі).

Змінюємо колір прямокутника, натиснувши інструмент  ***Fill Color*** та вибравши певний колір.



- **Створюємо кульку:**

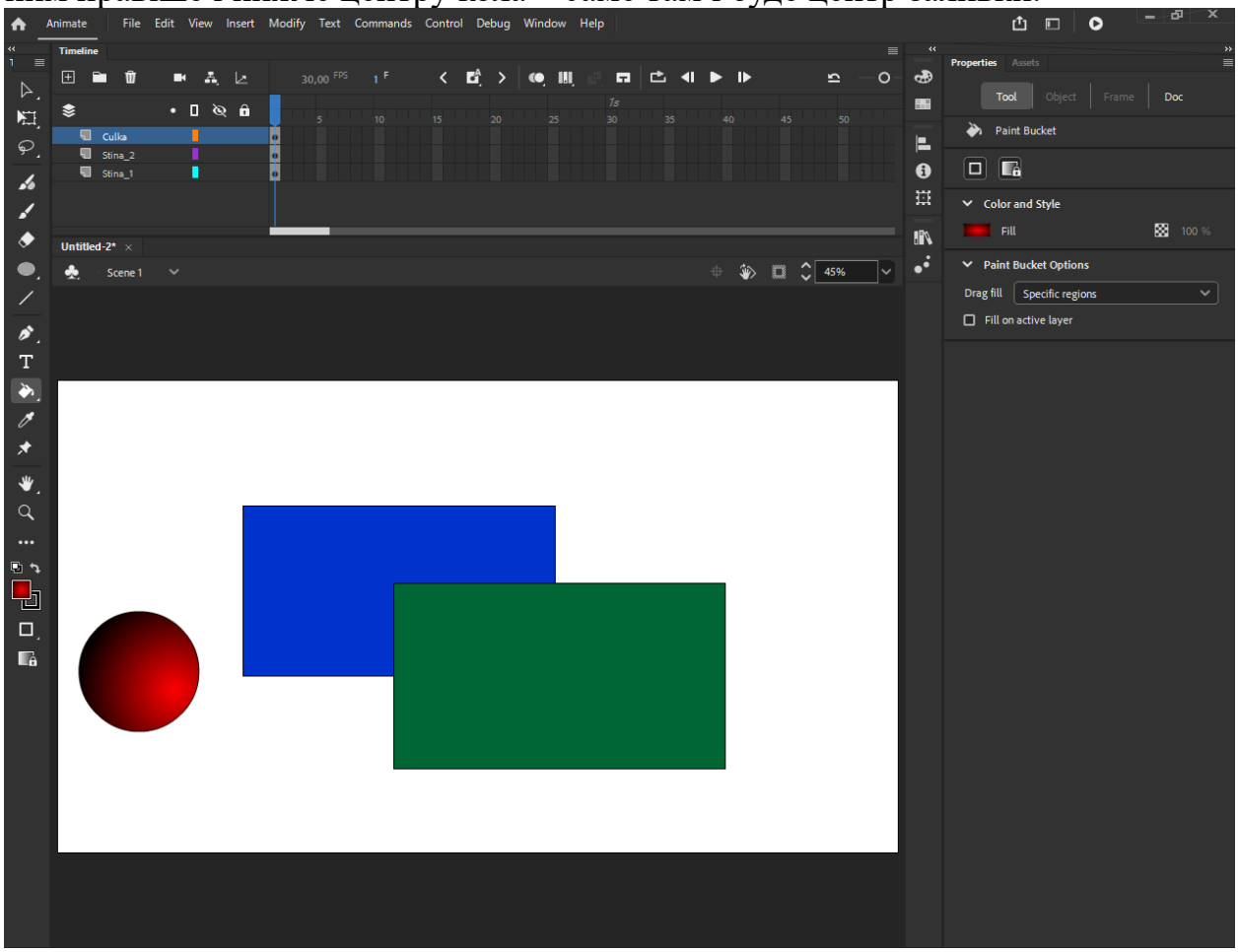
Додаємо ще один шар за допомогою інструмента  .
Змінюємо назву шару ***Layer_3*** на Culka.

За допомогою інструменту  ***Rectangle Tool*** *(натискаємо на нього мишею і тримаємо поки не розкриється вміст інструмента),* вибираємо інструмент  ***Oval Tool*** і з натиснутою на клавіатурі клавішею ***Shift*** (так буде малюватися коло) малюємо зліва від прямокутників коло діаметром приблизно в половину висоти прямокутника.

Вибираємо інструмент  ***Selection Tool*** і двічі клікаємо мишею в середині кола – виділяємо його. Якщо потрібно, можна відкоригувати розташування кола (за допомогою миші або стрілок на клавіатурі).

Змінюємо колір кола, натиснувши інструмент  ***Fill Color*** та вибравши один з кольорів (з центральним градієнтом) знизу палітри.

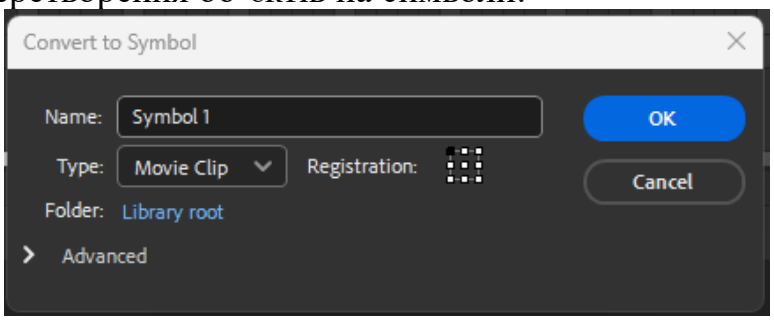
Змінимо центр заливки кола (щоб у подальшому було видно, як обертається кулька). Вибираємо інструмент  ***Paint Bucket Tool*** і клікаємо ним правіше і нижче центру кола – саме там і буде центр заливки.



3. Перетворюємо об’єкти на символи

Ми створили три об’єкти, кожен з яких розташований на окремому шарі. Якщо клікнути лівою кнопкою миші на назву шару, то виділиться той об’єкт, який на ньому знаходиться.

Клікнемо по шару ***Culka*** (виділимо кульку). Далі вибираємо у верхньому меню ***Modify => Convert to Symbol*** або натисніть на клавіатурі клавішу ***F8.*** З’явиться вікно перетворення об’єктів на символи.



Переконайтеся, що параметри такі самі, як на зображенні і натисніть кнопку ***Ok***. Параметр ***Movie Clip*** означає, що об’єкт прийматиме участь в анімації.

Якщо виконали все правильно, навколо об’єкта з’явиться блакитна

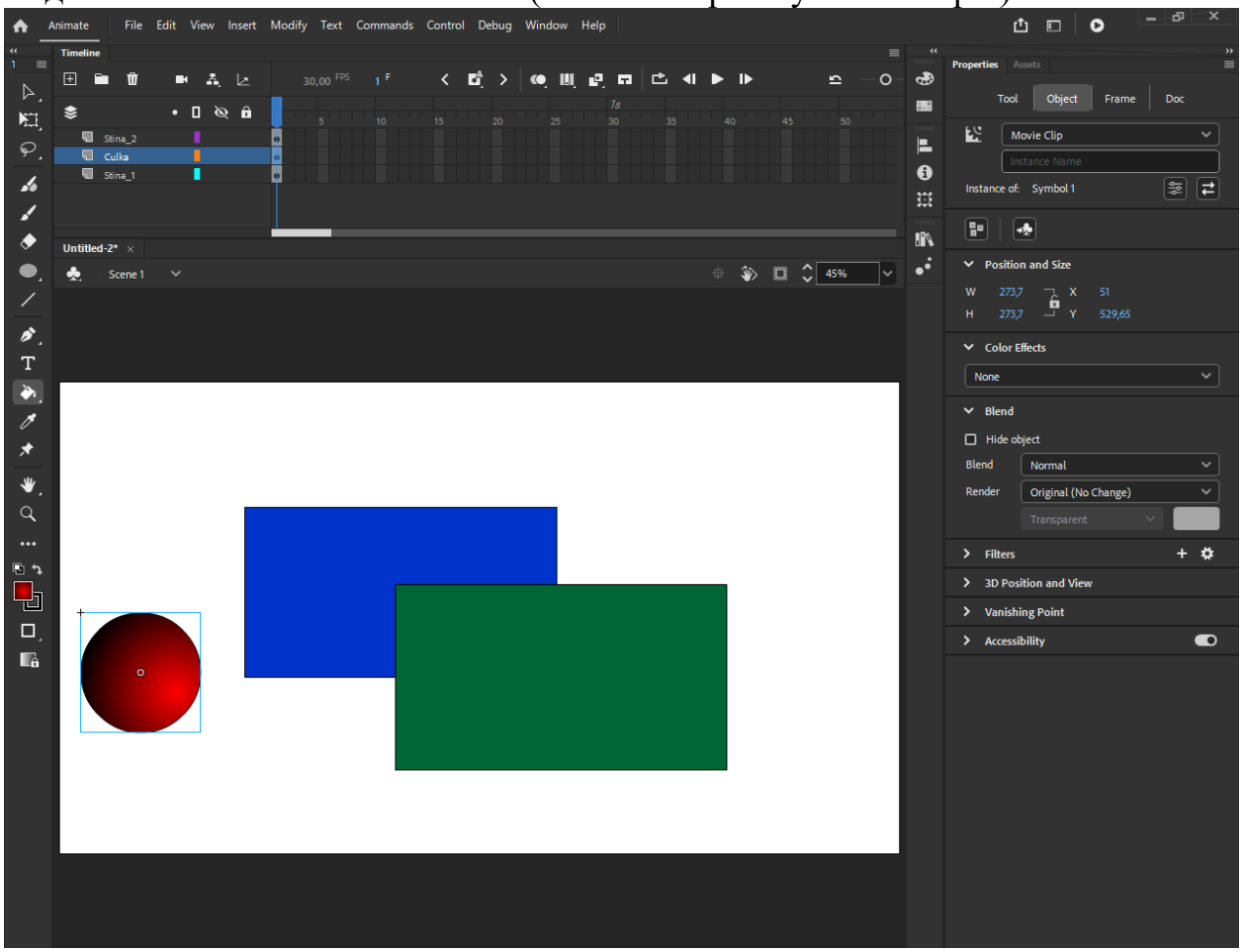
рамка. Вона буде відображатися завжди, коли виділено об'єкт.

Перетворіть інші об'єкти на символи – виконайте останні дії для об'єктів на інших шарах.

- ***Змінимо порядок розміщення шарів.***

Порядок розташування шарів по вертикалі грає важливу роль: чим вище розташований шар, тим нібито ближче до нас знаходиться об'єкт (символ).

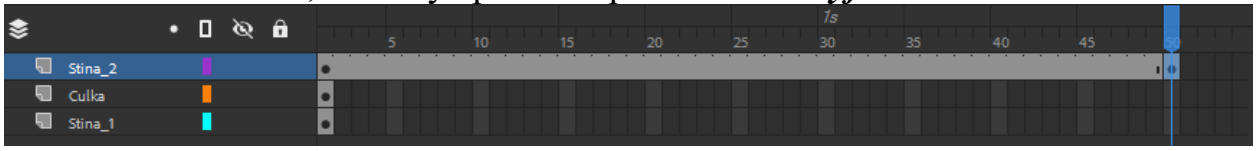
Нам необхідно, щоб кулька прокочувалася між стінками, отже шар ***Culka*** повинен знаходитись між шарами ***Stina_1*** і ***Stina_2***. Одна стінка повинна були ближче до нас, ніж інша, тому шар ***Stina_2*** буде вище ніж ***Stina_1***. Зробимо це за допомогою лівої кнопки миші (шляхом перетягування шарів).



4. Створюємо анімацію

Анімація триває певний час, який регламентується часовою шкалою Timeline (кількістю кадрів). Нехай наша анімація триває 50 кадрів (майже 2 секунди). Це означає, що усі об'єкти повинні відображатися протягом цього часу.

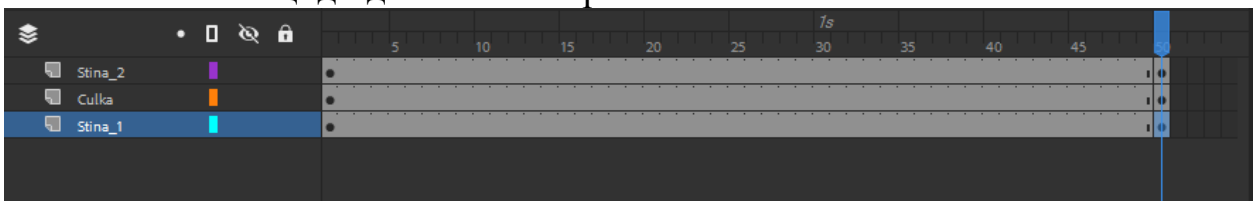
На шарі ***Stina_2*** клікнемо ***правою кнопкою миші*** на ***50 кадрі***. З'явиться ***контекстне меню***, в якому треба вибрати ***Insert Keyframe***.



Усі 50 кадрів шару ***Stina_2*** будуть зафарбовані одним кольором. Це

означає, що усі 50 кадрів мають одне й те саме зображення з однаковими його властивостями.

Виконайте ці дії для інших шарів.

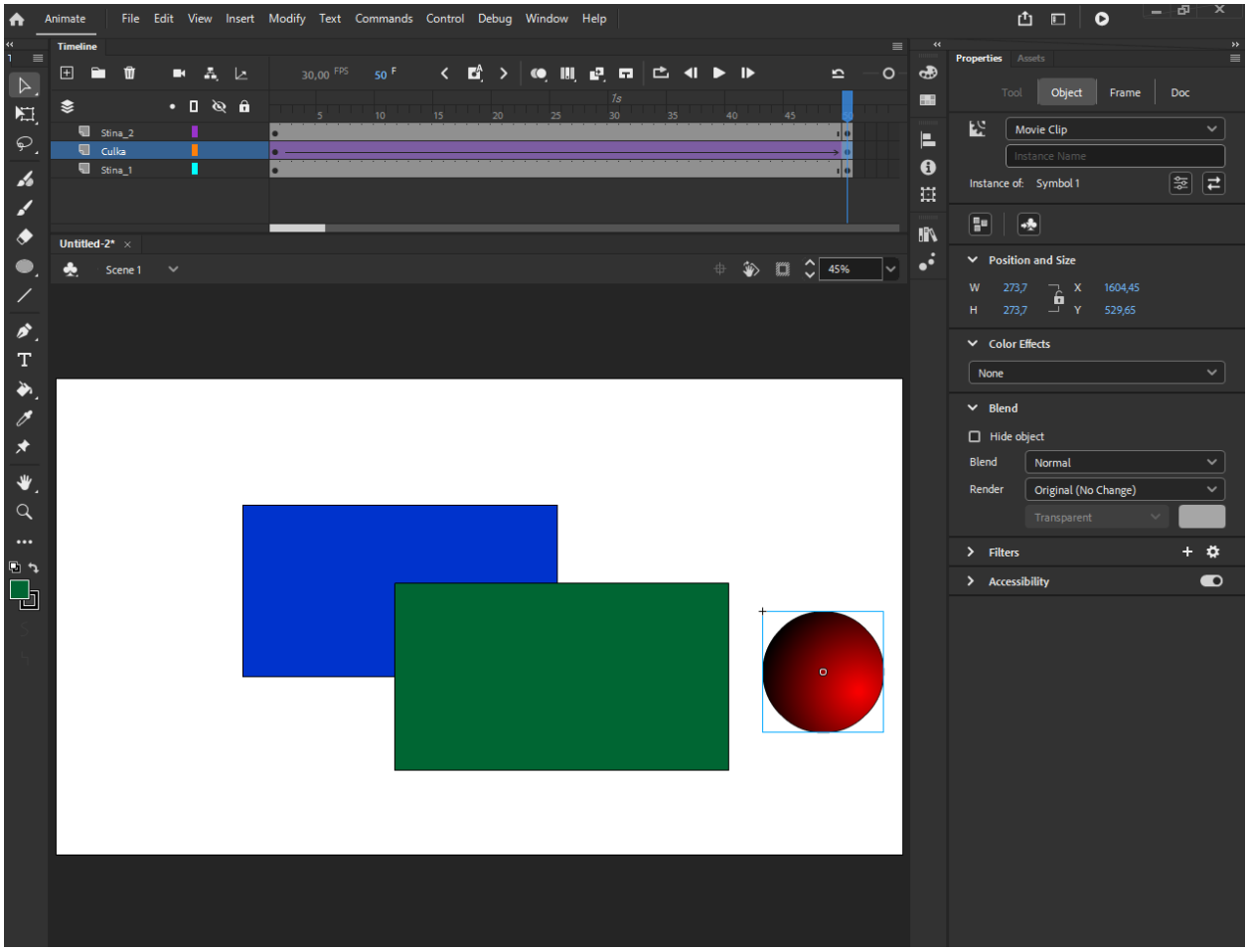


У нашій анімації буде рухатися тільки кулька, тому створюємо анімацію для шару з кулькою.

Клікніть ***правою кнопкою миші*** на ***першому кадрі*** шару ***Culka*** і з контекстного меню виберіть ***Create Classic Tween*** (створити класичну анімацію). Усі кадри між 1 і 50 змінять колір і на них з'явиться стрілка.

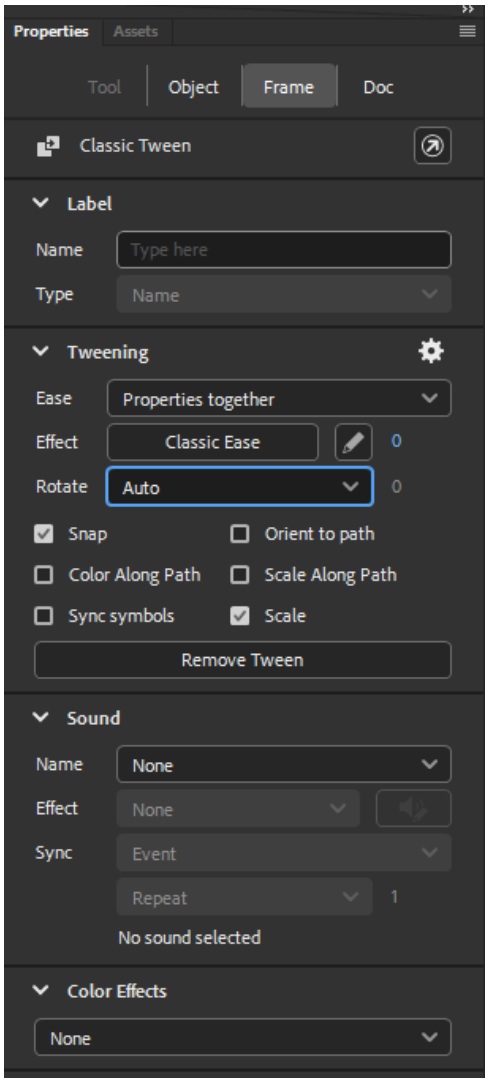
Кулька повинна котитися. Тому змінимо кінцеве положення кульки.

Вибираємо інструмент  ***Selection Tool***, клікнемо по ***50 кадру*** шару ***Culka*** (виділимо кульку в 50 кадрі). Перетягнемо кульку у нове положення – праворуч від прямокутників (стінок). Програма автоматично прорахує проміжні положення кульки (за найкоротшою траєкторією) в усіх кадрах з 2-го по 49-й.



5. Визначаємо властивості анімації

За допомогою інструменту  ***Selection Tool***, клікнемо по ***1-му кадру*** шару ***Culka*** (у цьому ключовому кадрі ми створили анімацію).



Панель властивостей ***Properties*** зміниться. Скористаємося нею для визначення параметрів анімації.

Встановимо, що кулька повинна обертатися за годинниковою стрілкою. Змінимо параметр ***Rotate*** з ***Auto*** на ***Clockwise*** (за годинниковою) або ***Counter Clockwise*** (проти годинникової) і ліворуч змінимо кількість обертів з **0** на **2** (кількість обертів кульки за весь час руху).

Тепер можна протестувати анімацію, натиснувши комбінацію клавіш ***Ctrl+Enter***.

6. Зробимо передню стінку прозорою.

За допомогою інструменту  ***Selection Tool***, клікнемо по ***1-му кадру*** шару ***Stina_2*** (тут найближча до нас стінка).

Панель властивостей ***Properties*** зміниться.

Змінимо параметр ***Color Effects*** з ***None*** на ***Alpha*** (прозорість об'єкту). Нижче на шкалі встановіть значення ***Alpha*** рівним ***50%***.

Теж саме зробіть для ***50-го кадру*** шару ***Stina_2.***

Протестуйте анімацію (***Ctrl+Enter***).

7. Збереження документу і публікація анімації.

Вибираємо у верхньому меню ***File => Save.*** Укажіть назву і місце розташування основного документу з анімацією. Тип файлу основного документу (який у подальшому можна буде редагувати) має розширення ****.fla***

Для публікації документа в інших форматах виберіть ***File => Publish Settings…*** У вікні, що з'явиться, виберіть необхідний формат (можна декілька одночасно) і натисніть спочатку ***Publish***, а потім ***OK***. Документи у вибраних форматах збережуться в тому самому місці, де знаходиться основний документ.

Завдання для самостійної роботи:

Спробуйте змінити створену просту анімацію руху кульки так, щоб вона котилася у зворотному напрямі.

Основи програмування в Adobe Animate – знайомство з ActionScript 3.0

1. Призначення та можливості ActionScript 3.0

ActionScript 3.0 (AS3.0) – це мова програмування, яка використовується в середовищі Adobe Animate для створення інтерактивних, анімованих і мультимедійних додатків. Вона є об'єктно-орієнтованою, що дозволяє створювати складні структури, повторно використовувати код, організовувати великі проєкти та забезпечувати високу продуктивність.

ActionScript 3.0 – потужний інструмент для програмування інтерактивних моделей у фізиці. Він дозволяє:

- Створювати та керувати об'єктами на сцені.
- Реалізовувати фізичні закони через змінні, оператори, функції.
- Реагувати на дії користувача через події.
- Організовувати складні симуляції через об'єктно-орієнтований підхід.
- Візуалізувати та аналізувати фізичні явища у реальному часі.

Опанування основ AS3.0 відкриває перед майбутнім учителем фізики широкі можливості для творчої реалізації, створення сучасних навчальних матеріалів і розвитку власних ІКТ-компетентностей.

Основні можливості ActionScript 3.0

- **Створення інтерактивних елементів**: кнопки, слайдери, поля введення, ігрові механіки, інтерактивні лабораторні роботи.
- **Керування анімацією**: програмне переміщення, обертання, масштабування об'єктів, зміна властивостей у реальному часі.
- **Обробка подій**: реагування на дії користувача (натискання миші, клавіатури), взаємодія між об'єктами.
- **Робота з мультимедіа**: керування звуком, відео, текстом, графікою.
- **Використання зовнішніх даних**: завантаження XML, робота з мережею, збереження та обробка інформації.
- **Підтримка 3D-анімації**: базова робота з 3D-об'єктами, їхнє позиціювання та обертання.
- **Висока продуктивність**: оптимізований рушій, який дозволяє створювати складні інтерактивні моделі без втрати швидкодії.

Переваги ActionScript 3.0 у навчанні фізики

- **Можливість моделювати фізичні процеси** (рух тіл, коливання, взаємодію об'єктів) у реальному часі.
- **Створення інтерактивних симуляцій** для лабораторних робіт, тестів, віртуальних експериментів.
- **Візуалізація абстрактних понять** через анімовані моделі, які реагують на дії користувача.
- **Розвиток алгоритмічного мислення** у здобувачів через створення власних моделей.

2. Синтаксис, змінні, оператори, події

Синтаксис ActionScript 3.0

AS3.0 має строгий синтаксис, подібний до JavaScript та інших мов сімейства C. Основні елементи:

- **Крапка з комою** (**;**) завершує кожен оператор.
- **Фігурні дужки** (**{ }**) позначають блоки коду.
- **Всі імена змінних, функцій, класів чутливі до регістру**.
- Коментарі можна писати після **двох косих рисок (*//*).**
- **У лапках (*""*)** завжди пишуть текстові рядки.

Приклад простого коду:

```
trace("Привіт, фізика!");
```

Змінні

Змінні – це іменовані області пам'яті для зберігання даних. Оголошуються за допомогою ключового слова var.

Приклад оголошення змінних:

```
var numApples:int = 4;
var guestName:String = "Іван";
var velocity:Number = 12.5;
```

У даному прикладі : int – цілі числа, Number – дійсні числа, String – рядки, Boolean – логічний тип (true/false).

Приклад зміни значення змінної:

```
var livesLeft:int = 3;
livesLeft = 2; // тепер значення 2
```

Приклад використання змінних у розрахунках:

```
var a:Number = 5;
var b:Number = 3;
var sum:Number = a + b; // sum = 8
trace(sum);
```

Приклад генерації випадкових чисел:

```
var randomValue:Number = Math.random() * 10; // від 0 до 10
```

Оператори

Оператори — це символи або ключові слова для виконання дій над змінними та значеннями.

- **Арифметичні**: +, -, *, /, %
- **Порівняння**: ==, !=, >, <, >=, <=
- **Логічні**: && (і), || (або), ! (не)
- **Присвоєння**: =, +=, -=, *=, /=

Приклад:

```
var x:int = 10;
var y:int = 5;
var result:int = x * y; // 50

if (result > 20) {
    trace("Результат більше 20");
}
```

Умовні оператори та цикли

Приклад використання умовного оператора if:

```
if (velocity > 0) {
    trace("Тіло рухається вперед");
} else {
    trace("Тіло стоїть або рухається назад");
}
```

Приклад використання циклу for:

```
for (var i:int = 0; i < 10; i++) {
    trace("Крок: " + i);
}
```

Приклад використання циклу while:

```
var t:int = 0;
while (t < 5) {
    trace("Час: " + t);
    t++;
}
```

Функції

Функції – це блоки коду, які можна викликати багаторазово.

Приклад використання функції:

```
function sum(a:Number, b:Number):Number {
    return a + b;
}
var total:Number = sum(3, 7); // total = 10
```

Події та обробники подій

Подія **–** це дія, яка відбувається у програмі (натискання кнопки, рух миші, натискання клавіші тощо).

Обробник події **–** це функція, яка виконується у відповідь на цю подію.

Основний синтаксис:

```
object.addEventListener(EventType.EVENT_NAME, handlerFunction);
function handlerFunction(event:EventType):void {
   // дії у відповідь на подію
}
```

Приклад: натискання кнопки

```
myButton.addEventListener(MouseEvent.CLICK, onClick);
function onClick(event:MouseEvent):void {
   trace("Кнопку натиснуто!");
}
```

Приклад: перетягування об'єкта

```
red_cat.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
red_cat.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
function startDragging(event:MouseEvent):void {
   red_cat.startDrag();
}
function stopDragging(event:MouseEvent):void {
   red_cat.stopDrag();
}
```

Власні (кастомні) події з параметрами

Можна створювати власні події та передавати дані між об'єктами.

```
// Відправлення події з параметром
var customEvent:Event = new Event("MY_EVENT");
dispatchEvent(customEvent);

// Обробка події
addEventListener("MY_EVENT", onMyEvent);

function onMyEvent(e:Event):void {
   // дії у відповідь
}
```

3. Створення інтерактивних елементів у фізичних моделях

Керування об'єктами на сцені

Усі графічні об'єкти на сцені в Adobe Animate є екземплярами класів (наприклад, MovieClip, Shape, Sprite). Кожен об'єкт має властивості (позиція, розмір, колір) і методи (рух, обертання, зміна стану).

Приклад: рух кулі по горизонталі

1. Додайте на сцену графічний об'єкт (наприклад, коло) і дайте йому на панелі Properties ім'я екземпляра ball.
2. Додайте код у перший ключовий кадр:

```
ball.x = 100; // початкова позиція
ball.y = 200;

function moveBall(event:Event):void {
   ball.x += 2; // рух праворуч
}
addEventListener(Event.ENTER_FRAME, moveBall);
```

Пояснення:

- *Event.ENTER_FRAME – подія, яка відбувається на кожному кадрі анімації (корисно для моделювання руху).*
- *ball.x += 2 – на кожному кадрі об'єкт зміщується на 2 пікселі праворуч.*

Моделювання рівномірного руху

Завдання: змоделювати рух тіла зі сталою швидкістю

Приклад коду для першого (ключового) кадру шару, на якому розміщено деякий об'єкт:

```
var velocity:Number = 5; // швидкість, пікселів за кадр

function moveObject(event:Event):void {
   ball.x += velocity;
}
addEventListener(Event.ENTER_FRAME, moveObject);
```

Якщо потрібно зупинити об'єкт при досягненні межі сцени, код можна доповнити:

```
function moveObject(event:Event):void {
   if (ball.x < stage.stageWidth - ball.width) {
      ball.x += velocity;
   } else {
      removeEventListener(Event.ENTER_FRAME, moveObject);
      trace("Тіло досягло межі сцени");
   }
}
```

Моделювання рівноприскореного руху

Завдання: змоделювати рух тіла з прискоренням

```
var velocity:Number = 0;
var acceleration:Number = 0.2;
function moveObject(event:Event):void {
   velocity += acceleration;
   ball.x += velocity;
}
addEventListener(Event.ENTER_FRAME, moveObject);
```

Пояснення: на кожному кадрі швидкість збільшується на величину прискорення, а координата – на величину нової швидкості.

Інтерактивне керування моделлю

Додавання кнопок для старту/зупинки руху

1. Додайте на сцену дві кнопки: startBtn і stopBtn.
2. Додайте код у перший ключовий кадр шару, де знаходяться кнопки:

```
startBtn.addEventListener(MouseEvent.CLICK, startMotion);
stopBtn.addEventListener(MouseEvent.CLICK, stopMotion);
function startMotion(e:MouseEvent):void {
   addEventListener(Event.ENTER_FRAME, moveObject);
}
function stopMotion(e:MouseEvent):void {
   removeEventListener(Event.ENTER_FRAME, moveObject);
}
```

Введення даних користувачем

Додавання текстового поля для введення швидкості

1. Додайте на сцену текстове поле inputSpeed (тип тексту Input на панелі властивостей).
2. Додайте поруч кнопку applyBtn.
3. Додайте код у перший ключовий кадр шару, де знаходиться кнопка:

```
applyBtn.addEventListener(MouseEvent.CLICK, setSpeed);
function setSpeed(e:MouseEvent):void {
   velocity = Number(inputSpeed.text);
}
```

Взаємодія з користувачем через мишу

Перетягування об’єкта

Приклад коду для об’єкта, який перетворено на символ з параметром Button:

```
ball.addEventListener(MouseEvent.MOUSE_DOWN, startDragBall);
ball.addEventListener(MouseEvent.MOUSE_UP, stopDragBall);
function startDragBall(e:MouseEvent):void {
   ball.startDrag();
}
function stopDragBall(e:MouseEvent):void {
   ball.stopDrag();
}
```

Моделювання фізичних явищ

Приклад коду для моделювання коливань маятника:

```
var angle:Number = 0;
var amplitude:Number = 50;
var centerX:Number = 200;
```

```
var centerY:Number = 200;
var frequency:Number = 0.05;
function oscillate(event:Event):void {
   angle += frequency;
   ball.x = centerX + amplitude * Math.sin(angle);
}
addEventListener(Event.ENTER_FRAME, oscillate);
```

***Пояснення*:** Math.sin(angle) створює гармонічні коливання, amplitude – амплітуда коливань, frequency – частота.

Відображення фізичних величин на сцені

Виведення поточного значення швидкості у динамічне текстове поле

1. Додайте текстове поле velocityText (тип тексту Dynamic на панелі властивостей).
2. Додайте у функцію руху:

```
velocityText.text = "Швидкість: " + velocity.toFixed(2) + " пікселів/кадр";
```

Використання класів для організації коду

ActionScript 3.0 підтримує об’єктно-орієнтоване програмування. Для складних моделей доцільно створювати власні класи.

Приклад: клас для моделі тіла

```
public class Body extends MovieClip {
   public var velocity:Number;
   public var acceleration:Number;
   public function Body(v:Number, a:Number) {
      velocity = v;
      acceleration = a;
      addEventListener(Event.ENTER_FRAME, move);
   }
   private function move(e:Event):void {
      velocity += acceleration;
      this.x += velocity;
   }
}
```

Використання на сцені:

```
var body:Body = new Body(2, 0.1);
addChild(body);
body.x = 100;
body.y = 200;
```

Обробка складних подій та взаємодія об’єктів

Власні події з параметрами

Для складних моделей (наприклад, взаємодія тіл) можна створювати власні події та передавати параметри між об’єктами.

```
// Створення власного класу події
public class CollisionEvent extends Event {
    public var energy:Number;
    public function CollisionEvent(type:String, energy:Number) {
        super(type);
        this.energy = energy;
    }
}
// Відправлення події
dispatchEvent(new CollisionEvent("COLLISION", 50));
// Обробка події
addEventListener("COLLISION", onCollision);
function onCollision(e:CollisionEvent):void {
    trace("Енергія зіткнення: " + e.energy);
}
```

4. Практичні приклади для фізики

Модель вільного падіння

```
var g:Number = 0.98; // прискорення вільного падіння
var velocity:Number = 0;
var y0:Number = 50;
ball.y = y0;
function fall(event:Event):void {
    velocity += g;
    ball.y += velocity;
    if (ball.y > stage.stageHeight - ball.height) {
        ball.y = stage.stageHeight - ball.height;
        velocity = -velocity * 0.7; // відскок з втратою енергії
    }
}
addEventListener(Event.ENTER_FRAME, fall);
```

Модель гармонічних коливань (пружинний маятник)

```
var k:Number = 0.1; // жорсткість пружини
var m:Number = 1; // маса
var x0:Number = ball.x;
var v:Number = 0;
function spring(event:Event):void {
    var F:Number = -k * (ball.x - x0);
    var a:Number = F / m;
    v += a;
```

```
    v *= 0.98; // демпфування
    ball.x += v;
}
addEventListener(Event.ENTER_FRAME, spring);
```

Візуалізація електричного поля точкового заряду

```
for (var i:int = 0; i < 360; i += 20) {
    var arrow:Arrow = new Arrow();
    arrow.rotation = i;
    arrow.x = charge.x + 50 * Math.cos(i * Math.PI / 180);
    arrow.y = charge.y + 50 * Math.sin(i * Math.PI / 180);
    addChild(arrow);
}
```

Тут Arrow – символ стрілки, charge – символ заряду.)

Практична робота 2. Створення анімації руху заданою траєкторією з елементами інтерактивності

Завдання: створити анімацію кочення кульки заданою траєкторією із можливістю керувати рухом кульки (запускати і зупиняти) за допомогою відповідних кнопок.

Загальна логіка дій:

- ✓ ***створити об'єкти, які будуть приймати участь в анімації, перетворити об'єкти на символи, встановити необхідні властивості для об'єктів;***
- ✓ ***створити просту анімацію руху для об'єкту (об'єктів);***
- ✓ ***для шару з анімацією створити додатковий шар з траєкторією, намалювати в ньому нерозривну і без перетинань лінію (траєкторію), у панелі властивостей для анімації встановити прив'язку об'єкта до траєкторії;***
- ✓ ***додати шар (зручно щоб він був першим – над усіма іншими шарами) для розміщення кнопок, намалювати кнопки і перетворити їх на символи, написати код для кнопок на першому ключовому кадрі шару з кнопками.***

1. Створюємо анімацію руху кульки

Запускаємо програму ***Adobe Animate***.

У стартовому вікні вибираємо параметр ***Full HD***.

У верхньому правому куті сцени вибираємо значення масштабу ***Fit in Window***.

На даний момент існує один шар ***Layer_1***, ***індикатор відтворення*** знаходиться на ***першому пустому ключовому кадрі*** даного шару.

Для зручності дамо назву шару у відповідності до створюваного об'єкта. Двічі клікнемо лівою кнопкою миші по назві шару ***Layer_1,*** пишемо Culka і натискаємо на клавіатурі клавішу ***Enter***.

За допомогою інструменту  ***Rectangle Tool*** *(натискаємо на нього мишею і тримаємо поки не розкриється вміст інструмента),* вибираємо інструмент  ***Oval Tool*** і з натиснутою на клавіатурі клавішею ***Shift*** (так буде малюватися коло) малюємо невелике коло.

Вибираємо інструмент  ***Selection Tool*** і двічі клікаємо мишею в середині кола – виділяємо його.

Змінюємо колір кола, натиснувши інструмент  ***Fill Color*** та вибравши один з кольорів (з центральним градієнтом) знизу палітри.

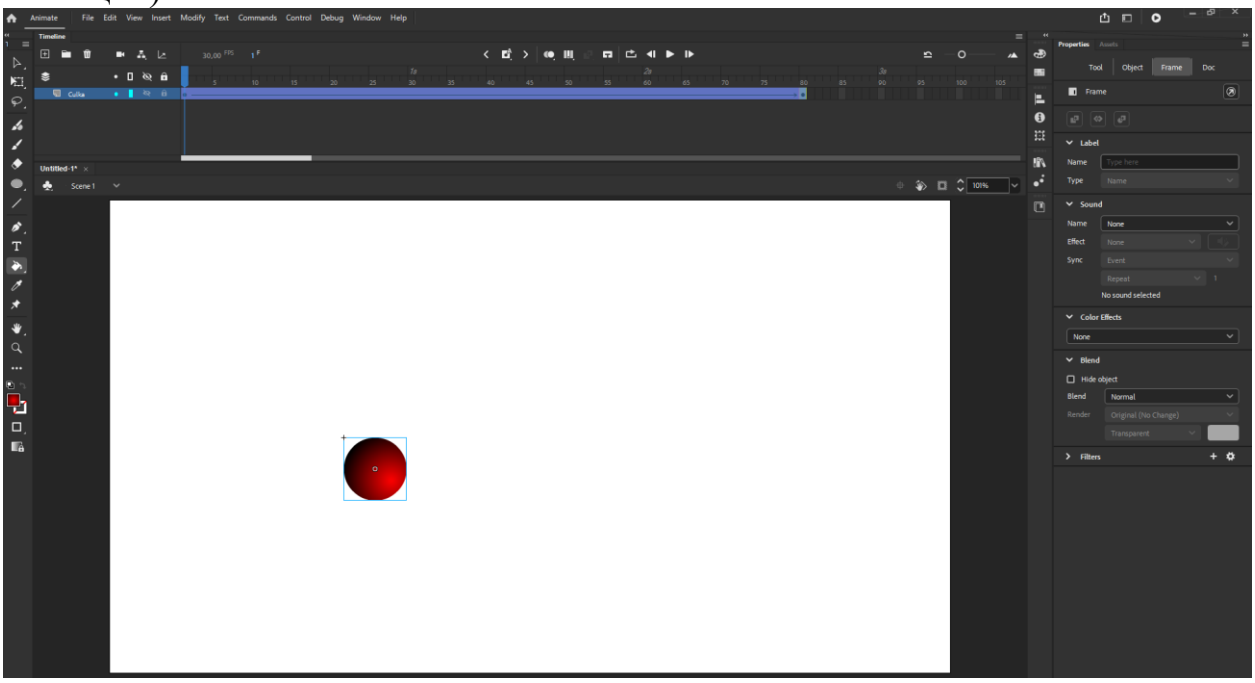
Змінимо центр заливки кола (щоб у подальшому було видно, як

обертається кулька). Вибираємо інструмент  **Paint Bucket Tool** і клікаємо ним правіше і нижче центру кола.

Клікнемо по шару ***Culka*** (виділимо кульку). Далі вибираємо у верхньому меню ***Modify => Convert to Symbol*** або натисніть на клавіатурі клавішу ***F8.*** З'явиться вікно перетворення об'єктів на символи. Встановлюємо параметр ***Movie Clip.***

Клікнемо ***правою кнопкою миші*** на ***80 кадрі***. З'явиться ***контекстне меню***, в якому треба вибрати ***Insert Keyframe***.

Клікніть ***правою кнопкою миші*** на ***першому кадрі*** шару ***Culka*** і з контекстного меню виберіть ***Create Classic Tween*** (створити класичну анімацію).



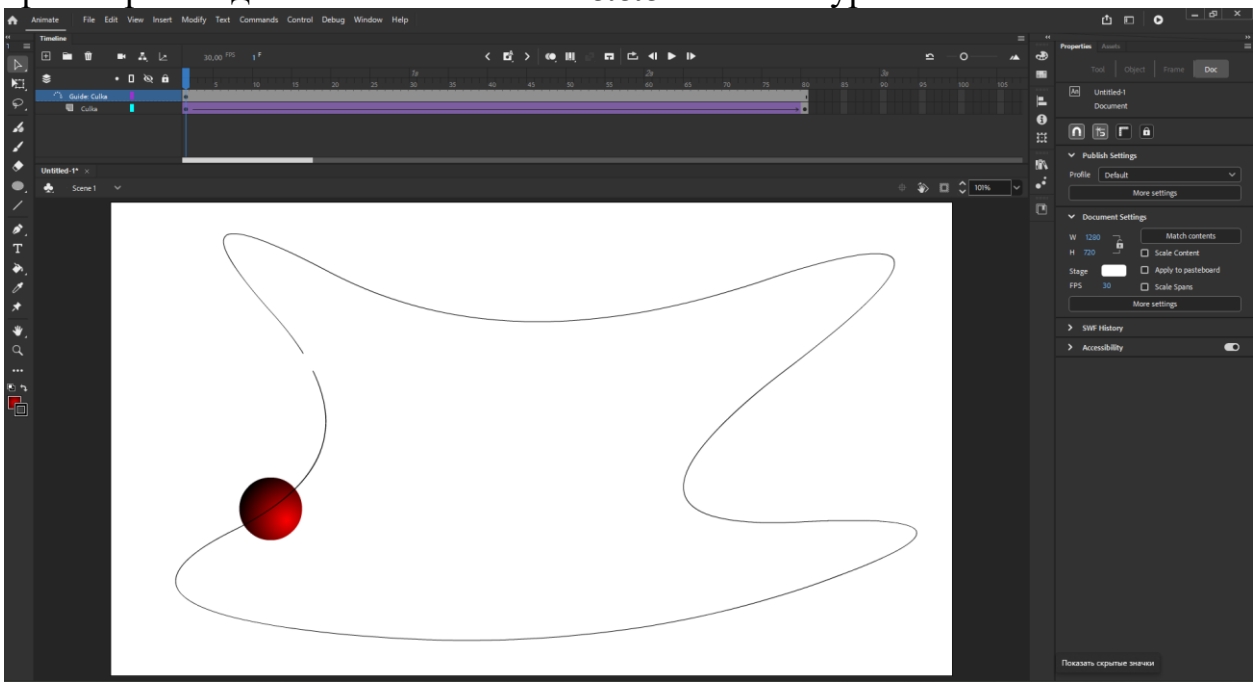
2. Створюємо траєкторію руху кульки

Клікнемо ***правою кнопкою миші*** по назві шару ***Culka*** і з контекстного меню вибираємо ***Add Classic Motion Guide***. Над шаром ***Culka*** з'явиться новий шар ***Guide: Culka***.

Намалюємо траєкторію (створимо траєкторію шляхом деформації лінії кола). Клікнемо по першому кадру шару ***Guide: Culka***, вибираємо інструмент  ***Oval Tool*** і малюємо великий овал на сцені. За допомогою інструменту  ***Selection Tool***, клікнемо всередині овалу і видалимо заливку шляхом натискання клавіши ***Delete*** на клавіатурі. Підводимо стрілку інструменту  ***Selection Tool*** майже до лінії овалу (біля стрілки повинна з'явитися дуга), затискаємо ліву кнопку миші і деформуємо овал. Робимо це з різних боків овалу так, щоб утворилася замкнена крива лінія.

Зробимо так, щоб траєкторія мала початок і кінець. За допомогою інструменту  ***Selection Tool*** виділяємо (обводимо) маленьку частинку

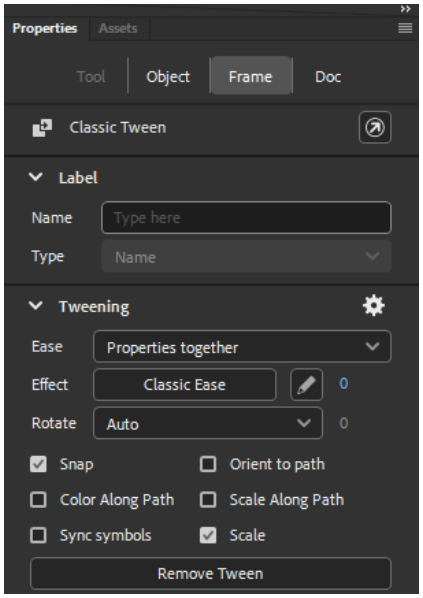
траєкторії і видаляємо її клавішею ***Delete*** на клавіатурі.



Клікніть ***лівою кнопкою миші*** на ***першому кадрі*** шару ***Culka*** і за допомогою інструменту  ***Selection Tool*** перетягніть кульку на початок траєкторії (центр кульки повинен збігтися з початком траєкторії).

Клікніть ***лівою кнопкою миші*** на ***80 кадрі*** шару ***Culka*** і за допомогою інструменту  ***Selection Tool*** перетягніть кульку на кінець траєкторії (центр кульки повинен збігтися з кінцем траєкторії).

Де початок, а де кінець траєкторії – ви визначаєте самостійно і довільно, як вам зручніше.



Знову клікніть ***лівою кнопкою миші*** на ***першому кадрі*** шару ***Culka*** за допомогою інструменту  ***Selection Tool***.

Зверніть увагу на верхню частину панелі властивостей ***Properties***. У ній містяться такі параметри: ***Snap***, ***Orient to path***, ***Color Along Path***, ***Scale Along Path***, ***Sync symbols***, ***Scale***. Ці параметри відповідають за поведінку об'єкту (кульки) під час руху траєкторією:

Snap – прив'язка об'єкта до траєкторії;

Orient to path – орієнтація розташування об'єкта по відношенню до траєкторії (схоже на те, як людина тримає рівновагу під час руху по похилій площині вгору або вниз);

Color Along Path – дозволяє об'єкту не лише рухатися вздовж заданого шляху, а й змінювати свій колір відповідно до кольору лінії шляху;

Scale Along Path – дозволяє об'єкту змінювати свій розмір (масштаб)

залежно від товщини лінії шляху;

Scale – це параметр, який визначає розмір об'єкта відносно його початкового розміру.

Нам достатньо буде залишити за замовченням відміченими параметри ***Snap*** та ***Scale***.

3. Додаємо інтерактивності (створюємо кнопки керування анімацією)

Додаємо ще один шар за допомогою інструмента  .

Змінюємо назву шару ***Layer_3*** на Knopki. На ньому будуть розміщені кнопки.

Створюємо кнопку, яка буде запускати анімацію.

На верхній панелі меню вибираємо ***Insert => New Symbol***, у віконці, що з'явиться, встановлюємо у параметрі ***Type*** значення ***Button***.

Робочий простір зміниться.

На сцені (посередині білого полотна) з'явиться позначка «+» – вона указує центральне положення зображення майбутньої кнопки.

Часова панель ***Timeline*** також зміниться – на ній міститиметься лише чотири кадри під назвами ***Up***, ***Over***, ***Down***, ***Hit***. Кількість шарів може бути довільною, але кількість кадрів в них залишатиметься незмінною.

Кадри, що будуть розміщені під назвою ***Up*** визначають вигляд кнопки, якою ми її бачитимемо до будь-яких дій (звичайний стан).

Кадри під назвою ***Over*** визначають вигляд кнопки, коли на неї наведена миша.

Кадри під назвою ***Down*** визначають вигляд кнопки, коли на неї натиснули.

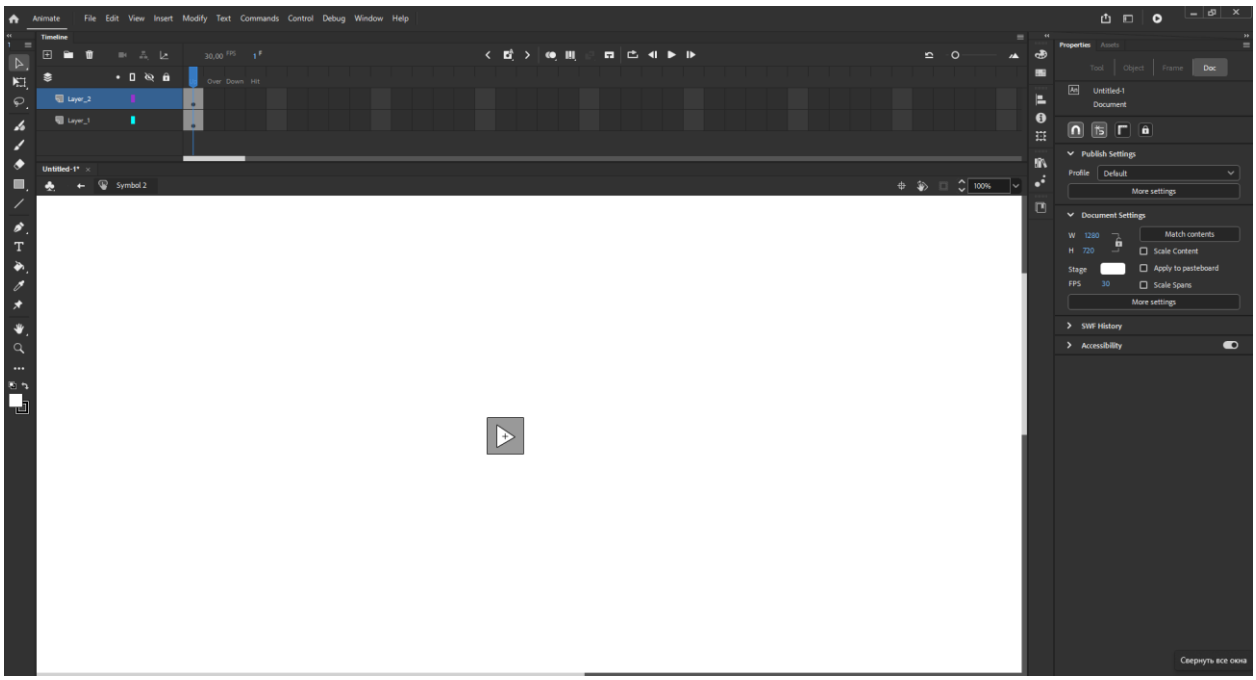
Hit – це спеціальний кадр, який визначає *активну область* кнопки, тобто ту зону, у межах якої відбувається сприйняття подій миші (наприклад, наведення курсора або клік). Іншими словами, кадр ***Hit*** задає форму і розмір "чутливої" області кнопки.

За допомогою інструменту  ***Selection Tool*** вибираємо перший кадр кнопки (***Up***) шару ***Layer_1.***

За допомогою інструменту  ***Rectangle Tool*** з натиснутою клавішею Shift малюємо невеликий квадрат так, щоб його центр збігся із позначкою центру сцени (якщо не збігається – перемістіть за допомогою стрілок клавіатури). Зафарбуйте квадрат сірим кольором.

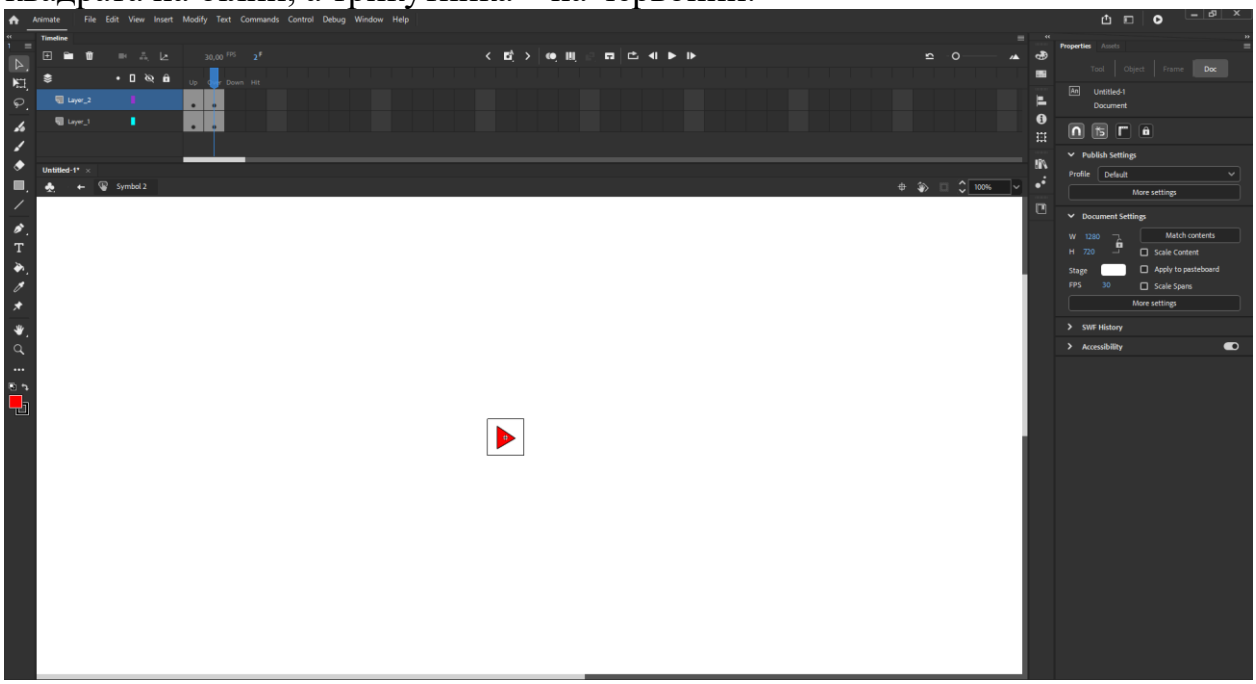
Додайте ще один шар ***Layer_2***. У першому кадрі цього шару поруч із квадратом намалюйте трикутник (вдвічі менший квадрату) і зафарбуйте його білим кольором. Перемістіть трикутник на центр сцени.

Отримали вигляд кнопки «пуск» у звичайному стані.



Змінимо вигляд кнопки, коли на неї наведена миша.

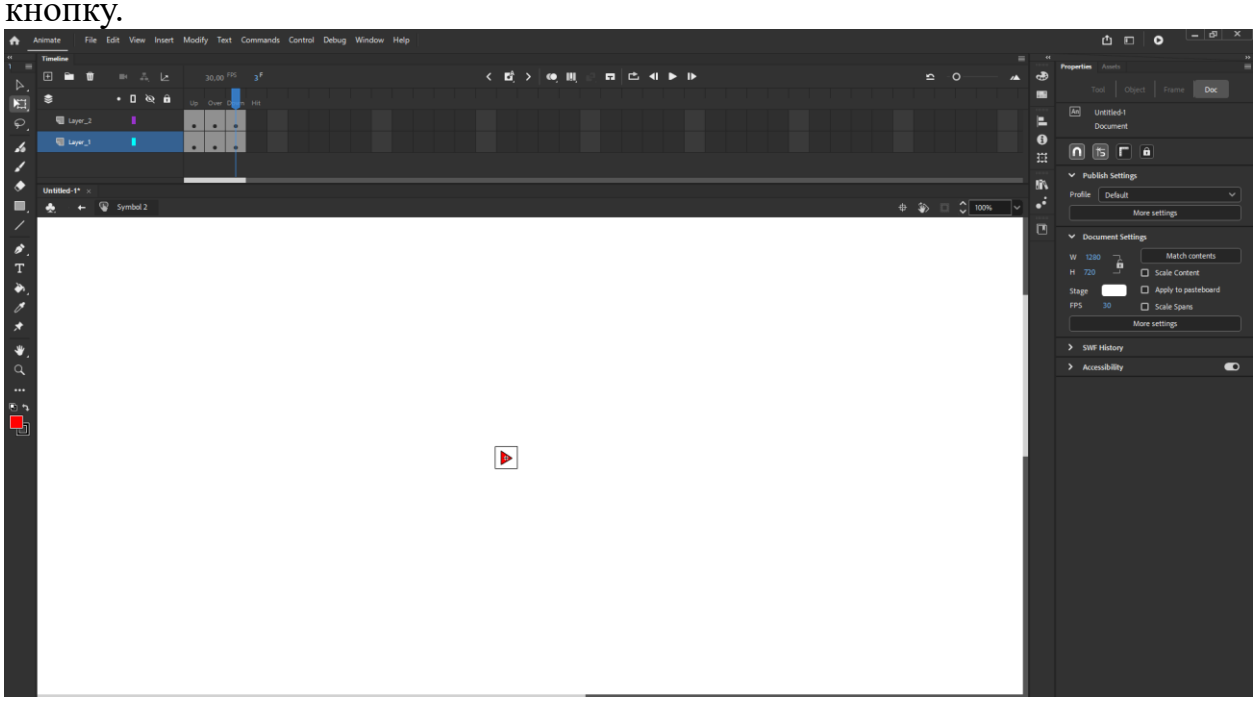
У кожному шарі під назвою кадрів ***Over*** клікаємо правою кнопкою миші по кадру і з контекстного меню вибираємо ***Insert Keyframe***. Отримаємо дублікати зображень квадрата і трикутника. Змінюємо їх колір заливки: квадрата на білий, а трикутника – на червоний.



Змінимо вигляд кнопки, коли на неї натиснули мишею.

У кожному шарі під назвою кадрів ***Down*** клікаємо правою кнопкою миші по кадру і з контекстного меню вибираємо Insert Keyframe. Отримаємо дублікати останніх зображень квадрата і трикутника. За допомогою інструменту  ***Selection Tool*** виділяємо (обводимо) кнопку. За допомогою

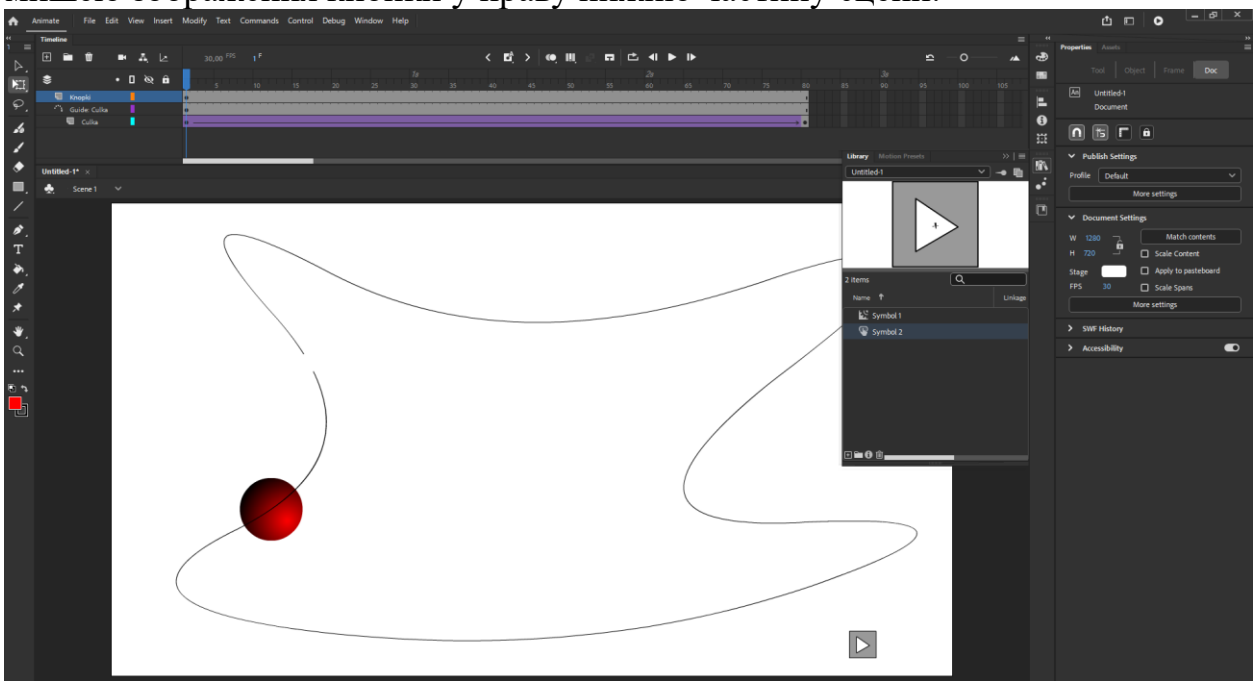
інструмента  ***Free Transform Tool*** і натиснутою клавішею ***Shift*** клавіатури змінюємо (зменшуємо вдвічі) розмір кнопки. Це створить ефект натискання на кнопку.



Розміщуємо кнопку на основній сцені.

Переходимо на основну сцену: натискаємо мишею на стрілочку ← біля назви ***Symbol 2*** (зліва над білим полотном сцени).

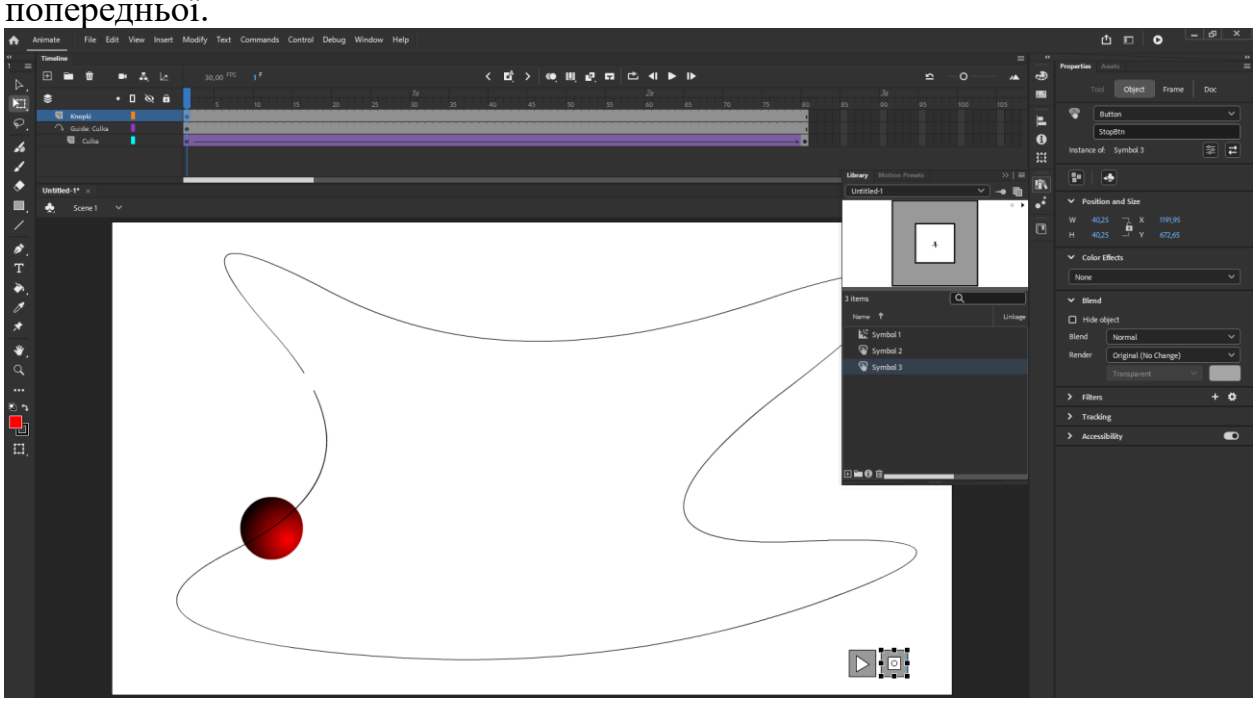
Входимо у бібліотеку символів: натискаємо на іконку  , що ліворуч від панелі властивостей ***Properties***. З віконця бібліотеки (Library) перетаскуємо мишею зображення кнопки у праву нижню частину сцени.



Дайте назву кнопці. Для цього клікніть мишею по зображенню кнопки на сцені

і у горі панелі властивостей ***Properties*** під параметром ***Button*** замість значення ***Instance Name*** напишіть ***PlayBtn***. Ця назва буде використана при написанні коду для кнопки.

Аналогічним способом створіть кнопку для зупинки анімації і дайте їй ім’я на панелі властивостей ***StopBtn***. Розташуйте цю кнопку праворуч від попередньої.



4. Пишемо код для кнопок

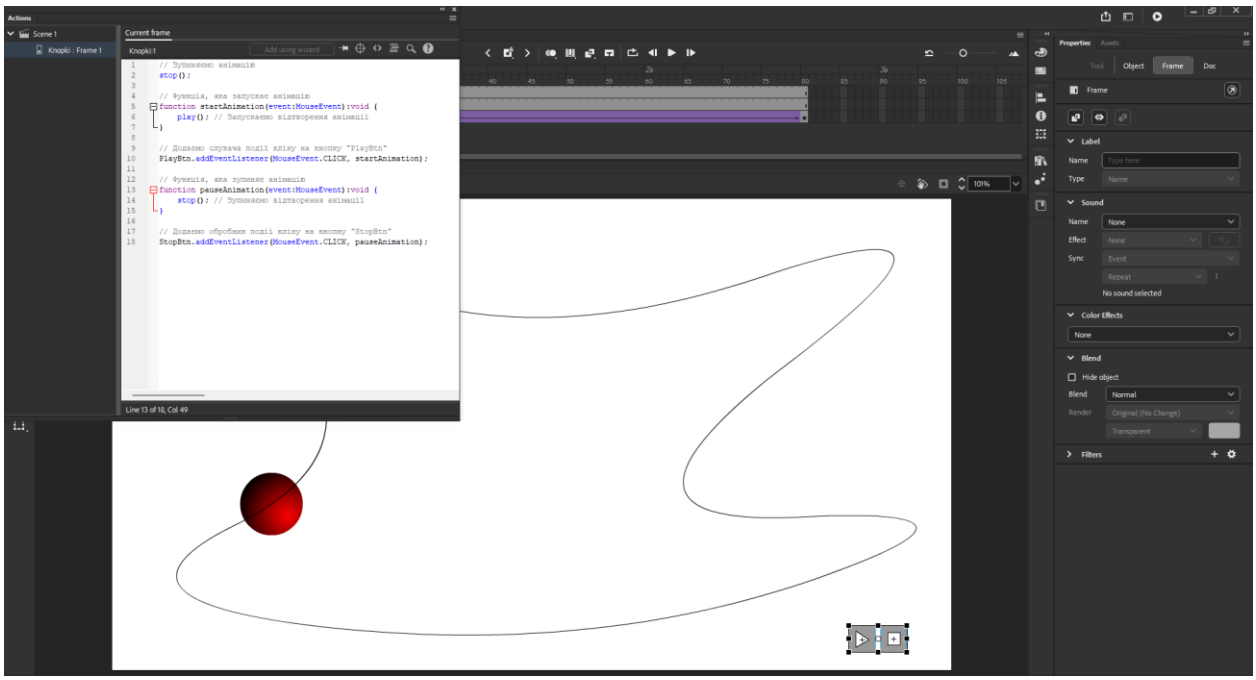
Обидві створені кнопки знаходяться на шарі ***Knopki***, тому код для кнопок будемо писати для першого (ключового) кадру цього шару.

За допомогою інструменту  ***Selection Tool*** вибираємо перший (***ключовий***) кадр шару ***Knopki.***

Натискаємо клавішу F9 на клавіатурі – викликаємо вікно ***Action Script***.

Пишемо такий код:

```
// Зупиняємо анімацію
stop();
// Функція для кнопки , яка запускає анімацію
function startAnimation(event:MouseEvent):void {
    play(); // Запускаємо відтворення анімації
}
// Додаємо слухача події кліку на кнопку " PlayBtn "
PlayBtn.addEventListener(MouseEvent.CLICK, startAnimation);
// Функція для кнопки, яка зупиняє анімацію
function pauseAnimation(event:MouseEvent):void {
    stop(); // Зупиняємо відтворення анімації
}
// Додаємо обробник події кліку на кнопку " StopBtn "
StopBtn.addEventListener(MouseEvent.CLICK, pauseAnimation);
```



Протестуємо створену анімацію (натиснути клавіши Ctrl + Enter).

Ви побачили, що анімація відтворюється не циклічно, вона зупиняється після того, як пройшло 80 кадрів і знову все повертається на 1 кадр.

Для того, щоб анімація відбувалась постійно (циклічно) можна для першого (ключового) кадру шару Culka (де створена анімація кульки) написати код, в якому по досягненню 80 кадру, вона буде відтворюватися з 2 кадру:

```
this.addEventListener(Event.ENTER_FRAME, checkFrame);

function checkFrame(e:Event):void {
    if (currentFrame == 80) {
        gotoAndPlay(2);
    }
}
```

Протестуйте анімацію. Збережіть та опублікуйте документ.

Завдання для самостійної роботи:

Спробуйте самостійно (по пам'яті) створити аналогічну анімацію, але щоб кулька котилася у зворотному напрямі.

Практична робота 3. Моделювання коливань нитяного маятника

***Завдання:** створити інтерактивну анімацію коливань нитяного маятника із демонстрацією змін величини і напряму вектору швидкості та висоти підняття тіла.*

Загальна логіка дій:

✓ *створити об'єкти, які будуть приймати участь в анімації (кожен об'єкт повинен бути на окремому шарі), перетворити об'єкти на символи, встановити необхідні властивості для об'єктів;*

✓ *реалізувати можливість керування відтворенням анімації через кнопки.*

1. Створюємо об'єкти

Запускаємо програму ***Adobe Animate***.

У стартовому вікні вибираємо параметр ***Full HD***.

У верхньому правому куті сцени вибираємо значення масштабу ***Fit in Window***.

На кожному окремому шарі створимо: фонові об'єкти (підвіс, підлогу), маятник (тіло на нитці), покажчик швидкості, покажчик висоти, кнопки пуску і зупинки анімації.

Створюємо фонові об'єкти.

На даний момент існує один шар ***Layer_1***, ***індикатор відтворення*** знаходиться на ***першому пустому ключовому кадрі*** даного шару.

Для зручності дамо назву шару у відповідності до створюваного об'єкта. Двічі клікнемо лівою кнопкою миші по назві шару ***Layer_1,*** пишемо ***fon*** і натискаємо на клавіатурі клавішу ***Enter***.

За допомогою інструменту  ***Line Tool*** зображуємо у верхній частині сцени підвіс, як показано на малюнку:

У нижній частині сцени малюємо такий самий об'єкт, але перевернутий і в чотири рази ширший за попередній – це буде підлога.

Створюємо маятник.

Додаємо ще один шар за допомогою інструмента  ліворуч на панелі Timeline.***Двічі клікаємо*** по назві шару ***Layer_2,*** пишемо ***mayatnyk*** і натискаємо на клавіатурі клавішу ***Enter***.

За допомогою інструменту  ***Line Tool*** малюємо вертикально вниз лінію від центру підвісу.

За допомогою інструменту  ***Rectangle Tool*** *(натискаємо на нього мишею і тримаємо поки не розкриється вміст інструмента),* вибираємо

інструмент  ***Oval Tool*** і з натиснутою на клавіатурі клавішею ***Shift*** (так буде малюватися коло) малюємо невелике коло.

Вибираємо інструмент  ***Selection Tool*** і двічі клікаємо мишею в середині кола – виділяємо його.

Змінюємо колір кола, натиснувши інструмент  ***Fill Color*** та вибравши один з кольорів (з центральним градієнтом) знизу палітри.

Розміщуємо отриману кульку на вільному кінці нитки.

Клікнемо по назві шару ***mayatnyk*** (виділимо маятник). Далі вибираємо у верхньому меню ***Modify => Convert to Symbol*** або натискаємо на клавіатурі клавішу ***F8.*** З'явиться вікно перетворення об'єктів на символи. Встановлюємо параметр ***Movie Clip*** і натискаємо OK.

Встановимо точку відліку, відносно якої буде обертатися маятник, на верхній край нитки. Для цього виділимо маятник (клікнемо по ньому) за допомогою інструменту  ***Free Transform Tool***. Посередині нитки побачимо білий кружечок, який і є точкою відліку. Перемістимо його вгору (до точки кріплення маятника на підвісі) за допомогою миші.

Якщо цього не зробити, маятник буде обертатися зверху і знизу одночасно відносно центру, а цього не повинно бути через те, що верхній край нитки закріплено на підвісі.

Створюємо вектор швидкості.

Додаємо ще один шар за допомогою інструмента  ліворуч на панелі Timeline. ***Двічі клікаємо*** по назві шару ***Layer_3,*** пишемо ***vektorspeed*** і натискаємо на клавіатурі клавішу ***Enter***.

За допомогою інструменту  ***Line Tool*** над кулькою малюємо стрілку направлену ліворуч.

Клікнемо по назві шару ***vektorspeed*** (виділимо стрілку). Далі вибираємо у верхньому меню ***Modify => Convert to Symbol*** або натискаємо на клавіатурі клавішу ***F8.*** З'явиться вікно перетворення об'єктів на символи. Встановлюємо параметр ***Movie Clip*** і натискаємо OK.

Встановимо точку відліку, відносно якої буде змінювати свою довжину стрілка. Для цього виділимо стрілку (клікнемо по ній) за допомогою інструменту  ***Free Transform Tool***. Посередині стрілки побачимо білий кружечок, який і є точкою відліку. Перемістимо його праворуч до початку стрілки за допомогою миші.

Створюємо позначення швидкості.

Додаємо ще один шар за допомогою інструмента  ліворуч на панелі Timeline. ***Двічі клікаємо*** по назві шару ***Layer_4,*** пишемо ***speed*** і натискаємо на клавіатурі клавішу ***Enter***.

На панелі інструментів вибираємо інструмент  ***Text Tool,*** на панелі властивостей ***Properties*** вибираємо шрифт ***Symbol***, встановлюємо ***Size*** рівним ***36*** і пишемо латинську літеру ***u*** (вона відобразиться як υ).

За допомогою інструменту  ***Line Tool*** над літерою υ малюємо маленьку стрілочку – позначення вектору швидкості.

Клікнемо по назві шару ***speed*** (виділимо літеру υ стрілку над нею). Далі вибираємо у верхньому меню ***Modify => Convert to Symbol*** або натискаємо на клавіатурі клавішу ***F8.*** З'явиться вікно перетворення об'єктів на символи. Встановлюємо параметр ***Movie Clip*** і натискаємо OK.

Створюємо покажчик висоти.

Додаємо ще один шар за допомогою інструмента  ліворуч на панелі Timeline. ***Двічі клікаємо*** по назві шару ***Layer_5,*** пишемо ***linevysota*** і натискаємо на клавіатурі клавішу ***Enter***.

За допомогою інструменту  ***Line Tool*** від кульки до підлоги малюємо двонапрямлену вертикальну стрілку.

Клікнемо по назві шару ***linevysota*** (виділимо стрілку). Далі вибираємо у верхньому меню ***Modify => Convert to Symbol*** або натискаємо на клавіатурі клавішу ***F8.*** З'явиться вікно перетворення об'єктів на символи. Встановлюємо параметр ***Movie Clip*** і натискаємо OK.

Встановимо точку відліку, відносно якої буде змінювати свою довжину стрілка. Для цього виділимо стрілку (клікнемо по ній) за допомогою інструменту  ***Free Transform Tool***. Посередині стрілки побачимо білий кружечок, який і є точкою відліку. Перемістимо його вгору на кінець стрілки за допомогою миші.

Створюємо позначення висоти.

Додаємо ще один шар за допомогою інструмента  ліворуч на панелі Timeline. ***Двічі клікаємо*** по назві шару ***Layer_6,*** пишемо ***vysota*** і натискаємо на клавіатурі клавішу ***Enter***.

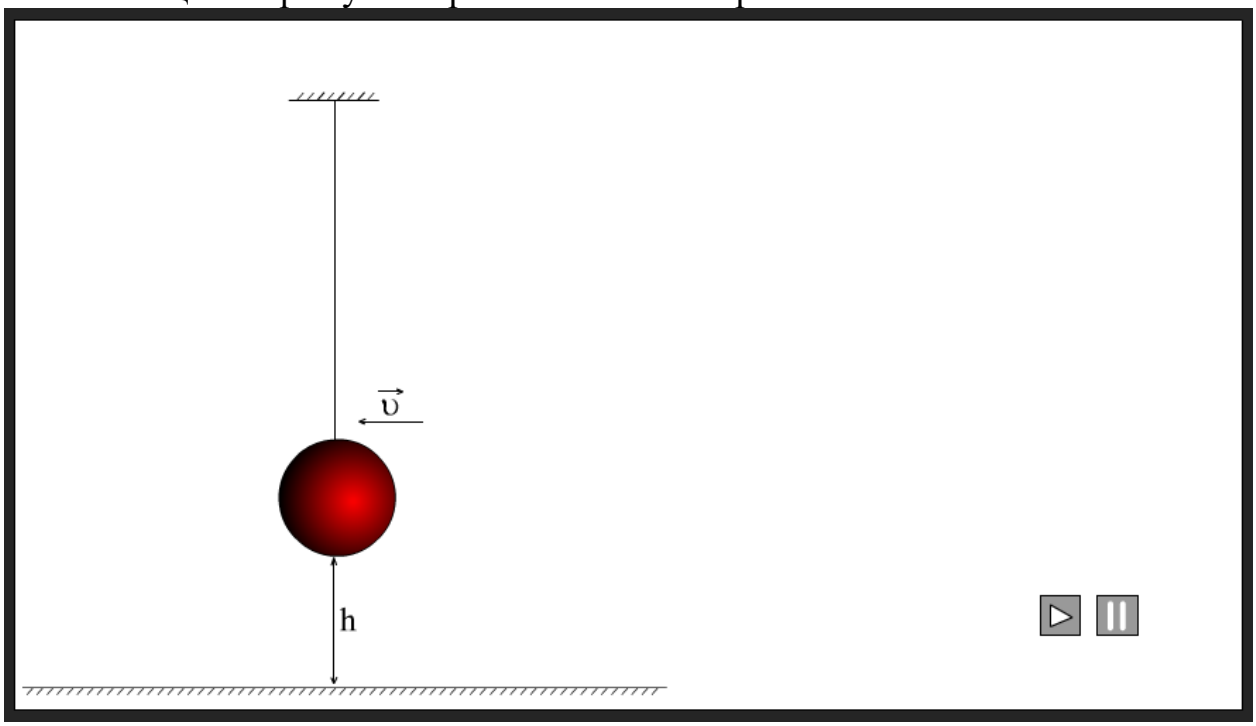
На панелі інструментів вибираємо інструмент  ***Text Tool,*** на панелі властивостей ***Properties*** вибираємо шрифт Times New Roman, встановлюємо ***Size*** рівним ***36*** і пишемо латинську літеру ***h*** і розміщуємо її біля центру і ліворуч стрілки.

Створюємо кнопки.

Додаємо ще один шар за допомогою інструмента  ліворуч на панелі Timeline. ***Двічі клікаємо*** по назві шару ***Layer_7,*** пишемо ***knopki*** і натискаємо на клавіатурі клавішу ***Enter***.

Створюємо кнопки пуск і стоп (див. попереднє практичне заняття). Даємо назву кнопкам ***PlayBtn*** і ***StopBtn*** відповідно.

На сцені отримуємо приблизно таке зображення:



Створюємо анімацію.

Створюємо анімацію коливань маятника.

У першому (ключовому) кадрі шару ***mayatnyk*** ми зобразили рівноважне положення маятника.

Клікаємо ***правою*** кнопкою миші на ***40*** кадрі шару ***mayatnyk*** і з контекстного меню вибираємо ***Insert Keyframe***. За допомогою інструменту ***Free Transform Tool*** повертаємо маятник ліворуч (приблизно на 45^0).

Клікаємо ***правою*** кнопкою миші на ***80*** кадрі шару ***mayatnyk*** і з контекстного меню вибираємо ***Insert Keyframe***. За допомогою інструменту ***Free Transform Tool*** повертаємо маятник у попереднє вертикальне положення.

Клікаємо ***правою*** кнопкою миші на ***120*** кадрі шару ***mayatnyk*** і з контекстного меню вибираємо ***Insert Keyframe***. За допомогою інструменту ***Free Transform Tool*** повертаємо маятник праворуч (приблизно на 45^0).

Клікаємо ***правою*** кнопкою миші на ***160*** кадрі шару ***mayatnyk*** і з контекстного меню вибираємо ***Insert Keyframe***. За допомогою інструменту ***Free Transform Tool*** повертаємо маятник у попереднє вертикальне положення.

Створюємо анімацію вектору швидкості.

У першому (ключовому) кадрі шару ***vektorspeed*** ми зобразили максимальне значення (довжини) вектору швидкості і напрям ліворуч. Під час анімації будуть змінюватися його координати, розміри і напрям.

Клікаємо ***правою*** кнопкою миші на ***39*** кадрі шару ***vektorspeed*** і з контекстного меню вибираємо ***Insert Keyframe***. Те ж саме робимо на ***40*** кадрі шару ***vektorspeed***. У ***39*** кадрі за допомогою інструменту ***Free Transform***

Tool зменшуємо довжину стрілки до мінімуму, а у ***40*** кадрі – перевертаємо стрілку на 180^0 і зменшуємо її розмір до мінімуму.

Клікаємо ***правою*** кнопкою миші на ***80*** кадрі шару ***vektorspeed*** і з контекстного меню вибираємо ***Insert Keyframe***. За допомогою інструменту  ***Free Transform Tool*** встановлюємо початковий розмір стрілки (але її напрям тепер протилежний).

Клікаємо ***правою*** кнопкою миші на ***119*** кадрі шару ***vektorspeed*** і з контекстного меню вибираємо ***Insert Keyframe***. Те ж саме робимо на ***120*** кадрі шару ***vektorspeed***. У ***119*** кадрі за допомогою інструменту  ***Free Transform Tool*** зменшуємо довжину стрілки до мінімуму, а у ***120*** кадрі – перевертаємо стрілку на 180^0 і зменшуємо її розмір до мінімуму.

Клікаємо ***правою*** кнопкою миші на ***160*** кадрі шару ***vektorspeed*** і з контекстного меню вибираємо ***Insert Keyframe***. За допомогою інструменту  ***Free Transform Tool*** встановлюємо початковий розмір стрілки.

У ключових кадрах шару ***vektorspeed*** (1, 40, 80 і 120) клікаємо правою кнопкою миші і з контекстного меню вибираємо ***Create Classic Tween*** (створити класичну анімацію).

Створюємо анімацію позначення швидкості.

У першому (ключовому) кадрі шару ***speed*** ми зобразили позначення швидкості. Під час анімації будуть змінюватися лише його координати.

Клікаємо ***правою*** кнопкою миші на ***40*** кадрі шару ***speed*** і з контекстного меню вибираємо ***Insert Keyframe***. За допомогою інструменту  ***Selection Tool*** переміщуємо позначення швидкості так, щоб воно було над стрілкою (вектором швидкості). Те ж саме робимо і для ***80***, ***120*** і ***160*** кадрів.

У ключових кадрах шару ***vektorspeed*** (1, 40, 80 і 120) клікаємо правою кнопкою миші і з контекстного меню вибираємо ***Create Classic Tween*** (створити класичну анімацію).

Створюємо анімацію покажчика висоти.

У першому (ключовому) кадрі шару ***linevysota*** ми зобразили покажчик висоти (двонапрямлену стрілку), яка має мінімальну довжину. Під час анімації будуть змінюватися її координати і розміри.

Клікаємо ***правою*** кнопкою миші на ***40*** кадрі шару ***linevysota*** і з контекстного меню вибираємо ***Insert Keyframe***. За допомогою інструменту  ***Free Transform Tool*** змінюємо висоту стрілки і її розташування так, щоб вона була під маятником. Те ж саме робимо і для ***80***, ***120*** і ***160*** кадрів.

У ключових кадрах шару ***vektorspeed*** (1, 40, 80 і 120) клікаємо правою кнопкою миші і з контекстного меню вибираємо ***Create Classic Tween*** (створити класичну анімацію).

Створюємо анімацію позначення висоти.

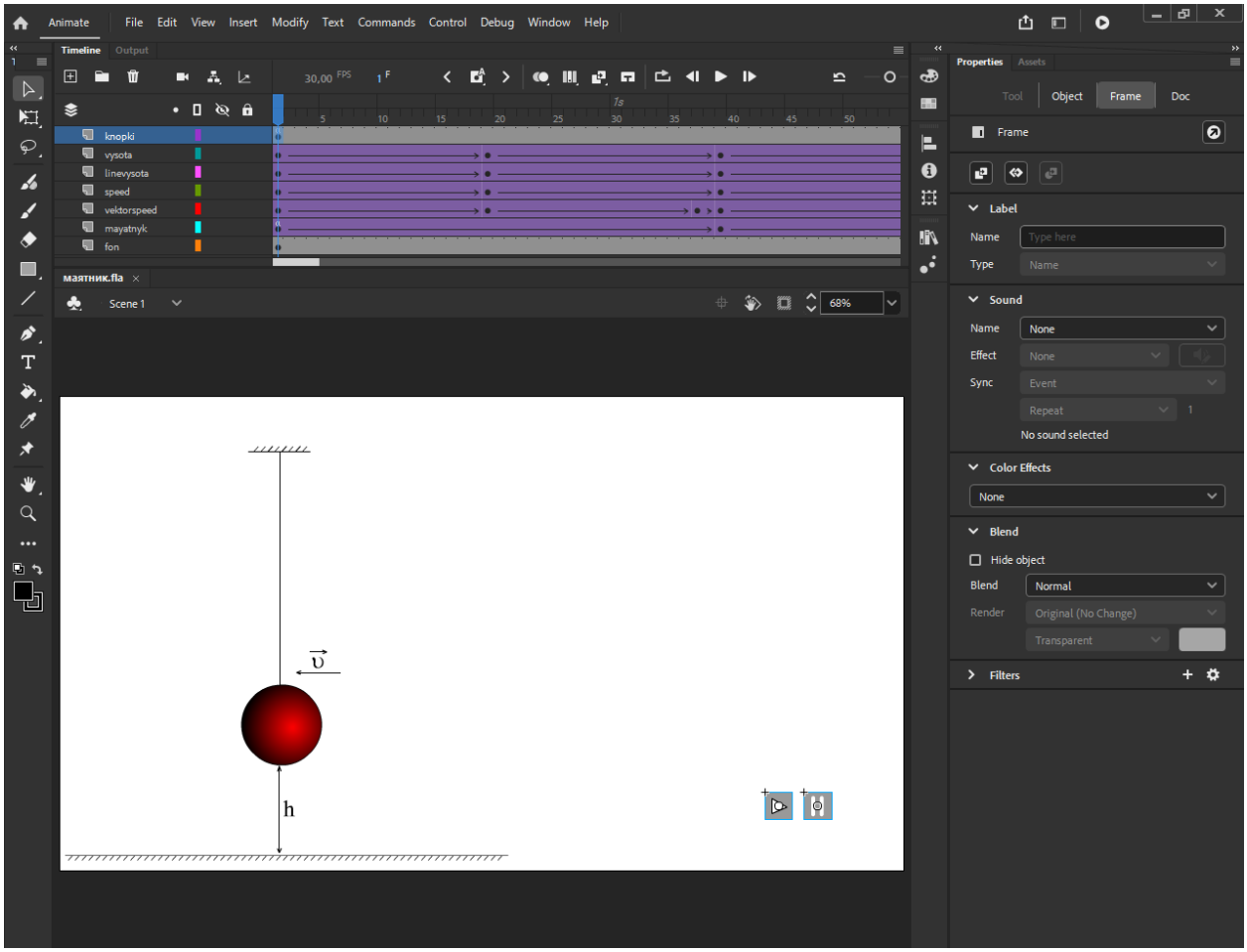
У першому (ключовому) кадрі шару ***vysota*** ми зобразили позначення висоти. Під час анімації будуть змінюватися лише його координати.

Клікаємо ***правою*** кнопкою миші на ***40*** кадрі шару ***vysota*** і з контекстного меню вибираємо ***Insert Keyframe***. За допомогою інструменту  ***Selection Tool*** переміщуємо позначення висоти так, щоб воно було біля середини двонапрямленої стрілки (покажчиком висоти). Те ж саме робимо і для ***80***, ***120*** і ***160*** кадрів.

У ключових кадрах шару ***vysota*** (1, 40, 80 і 120) клікаємо правою кнопкою миші і з контекстного меню вибираємо ***Create Classic Tween*** (створити класичну анімацію).

Зауваження.

Якщо в результаті тестування анімації позначення швидкості або висоти рухаються не синхронно із позначеннями швидкості або висоти, то додайте ключові кадри через кожні 20 кадрів, для більш точного коригування.



Додаємо код для кнопок та анімації.

Код для кнопок.

Клікаємо ***правою*** кнопкою миші по ***160*** кадру шару ***knopki*** і з

контекстного меню вибираємо ***Insert Keyframe***.(робимо це для того, щоб створені нами кнопки були видні протягом всіє анімації).

Клікаємо по ***першому*** (ключовому) кадру шару ***knopki***, натискаємо на клавіатурі клавішу ***F9*** і пишемо ***код для кнопок***:

```
stop();

// Функція, яка запускає анімацію
function startAnimation(event:MouseEvent):void {
    play(); // Запускаємо відтворення анімації
}

// Додаємо слухача події кліку на кнопку "startBtn"
startBtn.addEventListener(MouseEvent.CLICK, startAnimation);

// Функція, яка зупиняє анімацію
function pauseAnimation(event:MouseEvent):void {
    stop(); // Зупиняємо відтворення анімації
}

// Додаємо обробник події кліку на кнопку "stopBtn"
stopBtn.addEventListener(MouseEvent.CLICK, pauseAnimation);

import flash.events.MouseEvent;
import flash.external.ExternalInterface;
```

Код для анімації.

Коли ми запустимо тест анімації і натиснемо кнопку startBtn, анімація дійде до 160 кадру, перейде на 1 кадр і зупиниться. Для того щоб анімація виконувалася циклічно, треба зробити перехід з 160 кадру на 2 кадр. Зробимо це за допомогою коду.

Клікаємо по ***першому*** (ключовому) кадру шару ***mayatnyk***, натискаємо на клавіатурі клавішу ***F9*** і пишемо ***код для анімації***:

```
this.addEventListener(Event.ENTER_FRAME, checkFrame);

function checkFrame(e:Event):void {
    if (currentFrame == 160) {
        gotoAndPlay(2);
    }
}
```

Протестуйте анімацію. Збережіть та опублікуйте документ.

Завдання для самостійної роботи:

Спробуйте додати до анімації позначення і вектор прискорення.

Практична робота 4. Створення інтерактивного конструктора

***Завдання**: створити інтерактивний конструктор «сніговик», в якому на сцені хаотично розташовані різні частини сніговика, а нам за допомогою миші треба його зібрати шляхом перетягування.*

Загальна логіка дій:

- ✓ *створити об'єкти, які будуть приймати участь в анімації (кожен об'єкт повинен бути на окремому шарі), перетворити об'єкти на символи, встановити необхідні властивості для об'єктів;*
- ✓ *написати код, який дозволяє переміщувати об'єкти при натиснутій на них кнопці миші й забороняє переміщування, коли кнопка миші відпущена;*
- ✓ *реалізувати можливість повернення об'єктів у попередні положення.*

Створюємо об'єкти (частини сніговика)

Запускаємо програму ***Adobe Animate***.

У стартовому вікні вибираємо параметр ***Full HD***.

У верхньому правому куті сцени вибираємо значення масштабу ***Fit in Window***.

На кожному окремому шарі будемо створювати частини сніговика: нижній сніговий ком, середній ком, голову, ліву і праву руки, відро, мітлу.

На даний момент існує один шар ***Layer_1***, ***індикатор відтворення*** знаходиться на ***першому пустому ключовому кадрі*** даного шару.

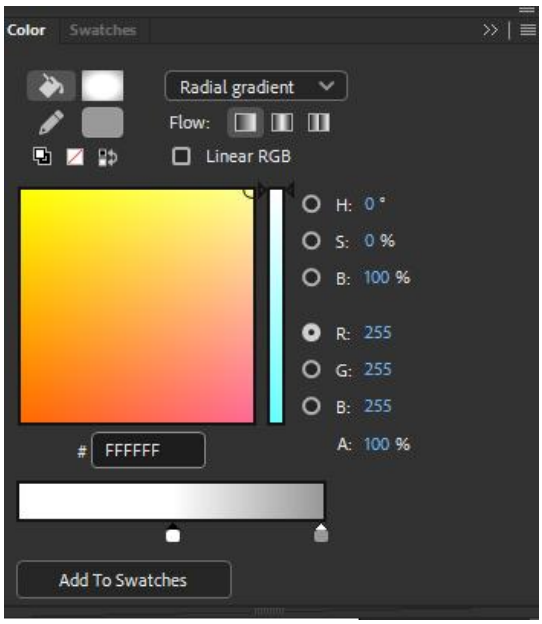
Для зручності дамо назву шару у відповідності до створюваного об'єкта. Двічі клікнемо лівою кнопкою миші по назві шару ***Layer_1,*** пишемо Niz і натискаємо на клавіатурі клавішу ***Enter***.

За допомогою інструменту  ***Rectangle Tool*** *(натискаємо на нього мишею і тримаємо поки не розкриється вміст інструмента),* вибираємо інструмент  ***Oval Tool*** і з натиснутою на клавіатурі клавішею ***Shift*** (так буде малюватися коло) малюємо невелике коло.

Вибираємо інструмент  ***Selection Tool*** і двічі клікаємо мишею в середині кола – виділяємо його.

Змінюємо колір кола, натиснувши інструмент  ***Fill Color*** та вибравши один з кольорів (з центральним градієнтом) знизу палітри.

Змінимо кольори градієнта ближче до кольорів снігу. На лівій частині панелі властивостей клікнемо інструмент  ***Color***, розкриється віконце редагування кольорів.



Знизу віконця знаходиться горизонтальна смужка, що регулює перехід від одного кольору до іншого:

Під смужкою містяться бігунки  (їх кількість залежить від кількості кольорів у заливці, їх можна додавати і видаляти), які відповідають за кольори, між якими відбувається перехід.

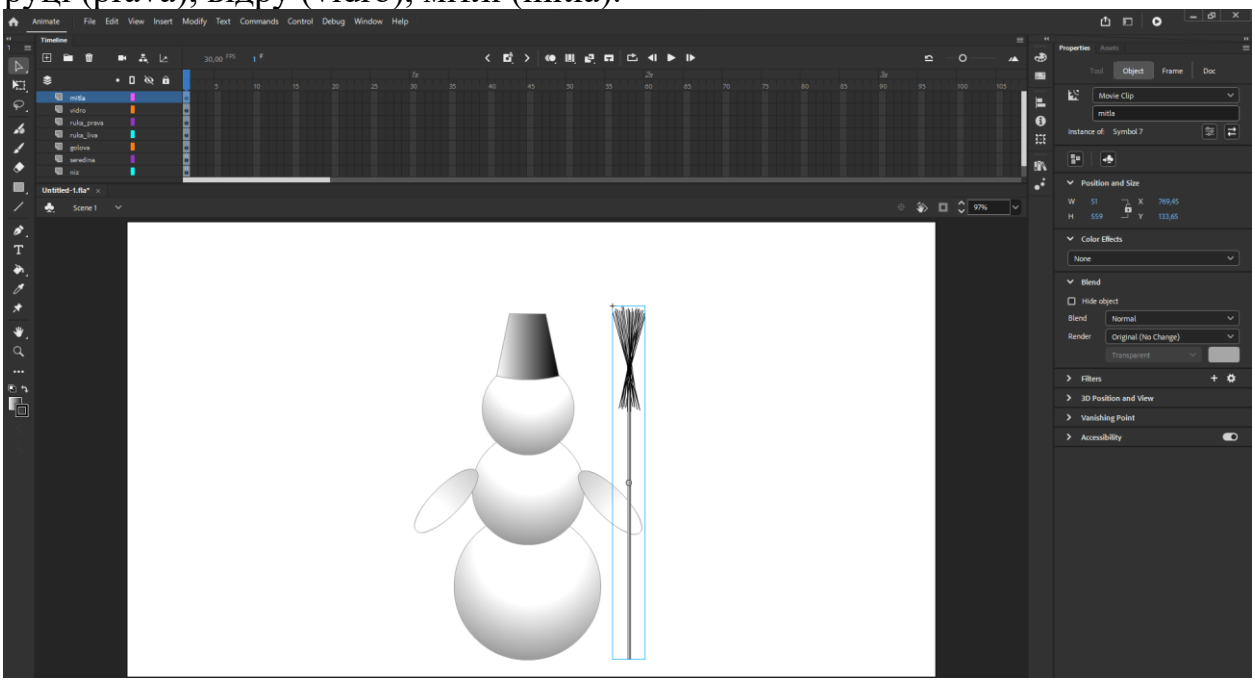
Двічі клікнемо мишею по лівому бігунку і виберемо білий колір. Потім двічі клікнемо мишею по правому бігунку і виберемо світло сірий колір.

Змінимо центр заливки кола (щоб додати реалістичності). Вибираємо інструмент  **Paint Bucket Tool** і клікаємо ним у верхній частині кола.

Клікнемо по шару Niz (виділимо об’єкт). Далі вибираємо у верхньому меню ***Modify => Convert to Symbol*** або натисніть на клавіатурі клавішу ***F8.*** З’явиться вікно перетворення об’єктів на символи. Встановлюємо параметр ***Movie Clip.***

Дамо назву об’єкту. Клікнемо по об’єкту на сцені й у панелі властивостей ***Properties*** під параметром ***Movie Clip*** замість значення ***Instance Name*** напишіть **niz**. Ця назва буде використана при написанні коду.

Аналогічно створюємо інші об’єкти на кожному окремому шарі, перетворюємо їх на символи, даємо назви об’єктам: нижньому кому (niz), середньому кому (seredina), верхньому кому (golova), лівій руці (liva), правій руці (prava), відру (vidro), мітлі (mitla).



Пишемо код для переміщення об'єктів

Для кожного ключового кадру відповідного шару з об'єктами напишемо код, який дозволятиме переміщувати ці об'єкти за допомогою миші. Для кожного ключового кадру код аналогічний і відрізняється лише назвою об'єкта, який в ньому міститься.

Для написання коду клікаємо по відповідному ключовому кадру мишею і натискаємо клавішу ***F9*** (виклик вікна Action Script).

Код для об'єкту* niz *(нижній сніговий ком)

```
//Перетворюємо об'єкт на кнопку
niz.buttonMode = true;

//Дозволяємо перетаскувати об'єкт при натиснутій на ньому кнопці миші
niz.addEventListener(MouseEvent.MOUSE_DOWN, startDragNiz);

// Забороняємо перетаскування об'єкту, коли кнопку миші відпустили
function startDragNiz(e:MouseEvent):void {
   niz.startDrag();
   stage.addEventListener(MouseEvent.MOUSE_UP, stopDragNiz);
}

// Видаляємо обробку подій, щоб не було впливу на інші обробники
function stopDragNiz(e:MouseEvent):void {
   niz.stopDrag();
   stage.removeEventListener(MouseEvent.MOUSE_UP, stopDragNiz);
}
```

Аналогічний код для об'єкту* seredina *(середній сніговий ком)

```
seredina.buttonMode = true;
seredina.addEventListener(MouseEvent.MOUSE_DOWN,
startDragseredina);
function startDragseredina(e:MouseEvent):void {
   seredina.startDrag();
   stage.addEventListener(MouseEvent.MOUSE_UP, stopDragseredina);
}
function stopDragseredina(e:MouseEvent):void {
   seredina.stopDrag();
   stage.removeEventListener(MouseEvent.MOUSE_UP, stopDragseredina);
}
```

Код для об'єкту* golova *(верхній сніговий ком)

```
golova.buttonMode = true;
golova.addEventListener(MouseEvent.MOUSE_DOWN, startDraggolova);
function startDraggolova(e:MouseEvent):void {
   golova.startDrag();
   stage.addEventListener(MouseEvent.MOUSE_UP, stopDraggolova);
```

```
}
function stopDraggolova(e:MouseEvent):void {
   golova.stopDrag();
   stage.removeEventListener(MouseEvent.MOUSE_UP, stopDraggolova);
}
```

Код для об'єкту* liva *(ліва рука)

```
liva.buttonMode = true;
liva.addEventListener(MouseEvent.MOUSE_DOWN, startDragliva);
function startDragliva(e:MouseEvent):void {
   liva.startDrag();
   stage.addEventListener(MouseEvent.MOUSE_UP, stopDragliva);
}
function stopDragliva(e:MouseEvent):void {
   liva.stopDrag();
   stage.removeEventListener(MouseEvent.MOUSE_UP, stopDragliva);
}
```

Код для об'єкту* prava *(права рука)

```
prava.buttonMode = true;
prava.addEventListener(MouseEvent.MOUSE_DOWN, startDragprava);
function startDragprava(e:MouseEvent):void {
   prava.startDrag();
   stage.addEventListener(MouseEvent.MOUSE_UP, stopDragprava);
}
function stopDragprava(e:MouseEvent):void {
   prava.stopDrag();
   stage.removeEventListener(MouseEvent.MOUSE_UP, stopDragprava);
}
```

Код для об'єкту* vidro *(відро)

```
vidro.buttonMode = true;
vidro.addEventListener(MouseEvent.MOUSE_DOWN, startDragvidro);
function startDragvidro(e:MouseEvent):void {
   vidro.startDrag();
   stage.addEventListener(MouseEvent.MOUSE_UP, stopDragvidro);
}
function stopDragvidro(e:MouseEvent):void {
   vidro.stopDrag();
   stage.removeEventListener(MouseEvent.MOUSE_UP, stopDragvidro);
}
```

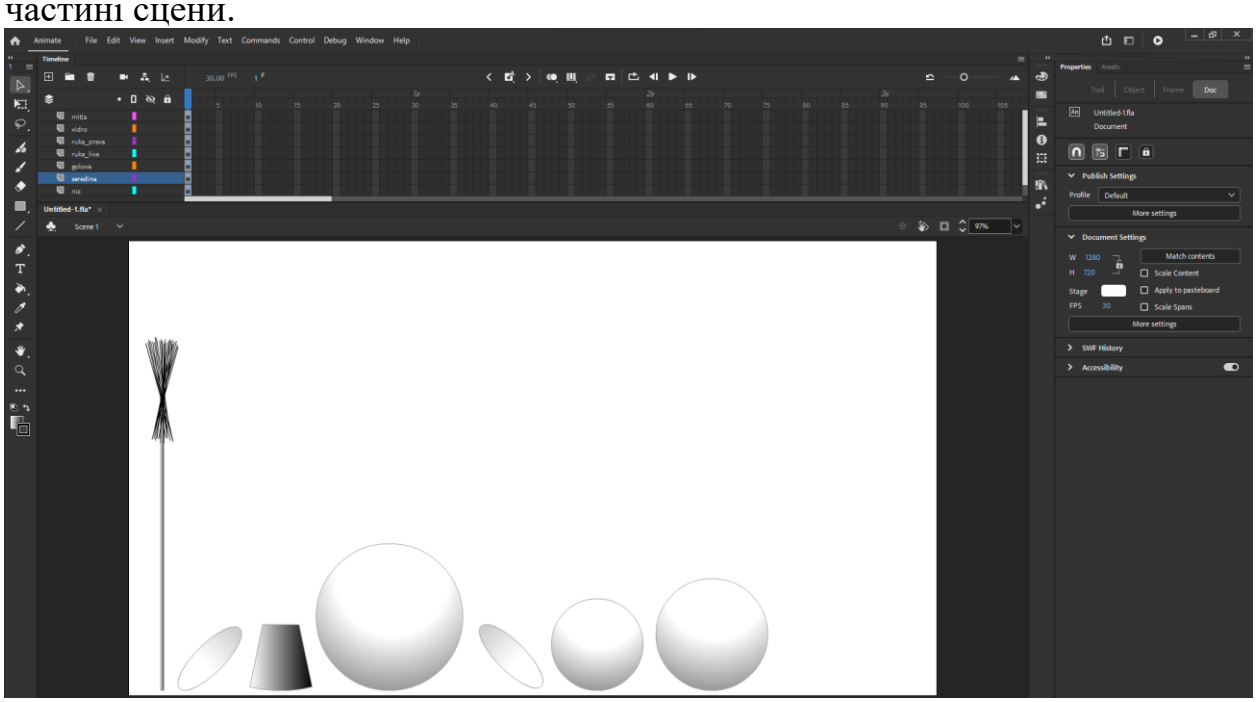
Код для об'єкту* mitla *(мітла)

```
mitla.buttonMode = true;
mitla.addEventListener(MouseEvent.MOUSE_DOWN, startDragmitla);
```

```
function startDragmitla(e:MouseEvent):void {
    mitla.startDrag();
    stage.addEventListener(MouseEvent.MOUSE_UP, stopDragmitla);
}
function stopDragmitla(e:MouseEvent):void {
    mitla.stopDrag();
    stage.removeEventListener(MouseEvent.MOUSE_UP, stopDragmitla);
}
```

Створюємо кнопку Reset

У нашому випадку сніговик зібраний. Розмістимо хаотично частини сніговика – просто пересуньте усі частини так, щоб вони знаходилися у нижній частині сцени.



Тепер, коли запустимо анімацію, потрібно буде зібрати сніговика.

Створимо кнопку, яка буде повертати усі частини сніговика у початкові положення.

Додаємо ще один шар (він повинен бути самим верхнім) і змінюємо його назву на knopka.

На верхній панелі меню вибираємо ***Insert => New Symbol***, у віконці, що з'явиться, встановлюємо у параметрі ***Type*** значення ***Button***. Натискаємо **ОК**.

За допомогою інструменту  ***Selection Tool*** вибираємо перший кадр кнопки (***Up***) шару ***Layer_1.***

За допомогою інструменту  ***Rectangle Tool*** малюємо невеликий широкий прямокутник так, щоб його центр збігся із позначкою центру сцени (якщо не збігається – перемістіть за допомогою стрілок клавіатури). Зафарбуйте прямокутник сірим кольором.

Додайте ще один шар ***Layer_2***. У першому кадрі цього шару за

допомогою інструмента  ***Text Tool*** пишемо на прямокутнику слово Reset і зафарбуйте його білим кольором. Розташуйте слово в середині прямокутника (на центрі сцени).

Отримали вигляд кнопки «Reset» у звичайному стані.

Змінимо вигляд кнопки, коли на неї наведена миша.

У кожному шарі під назвою кадрів ***Over*** клікаємо правою кнопкою миші по кадру і з контекстного меню вибираємо ***Insert Keyframe***. Отримаємо дублікати зображень прямокутника і слова. Змінюємо їх колір заливки: прямокутника на білий, а слова – на червоний.

Змінимо вигляд кнопки, коли на неї натиснули мишею.

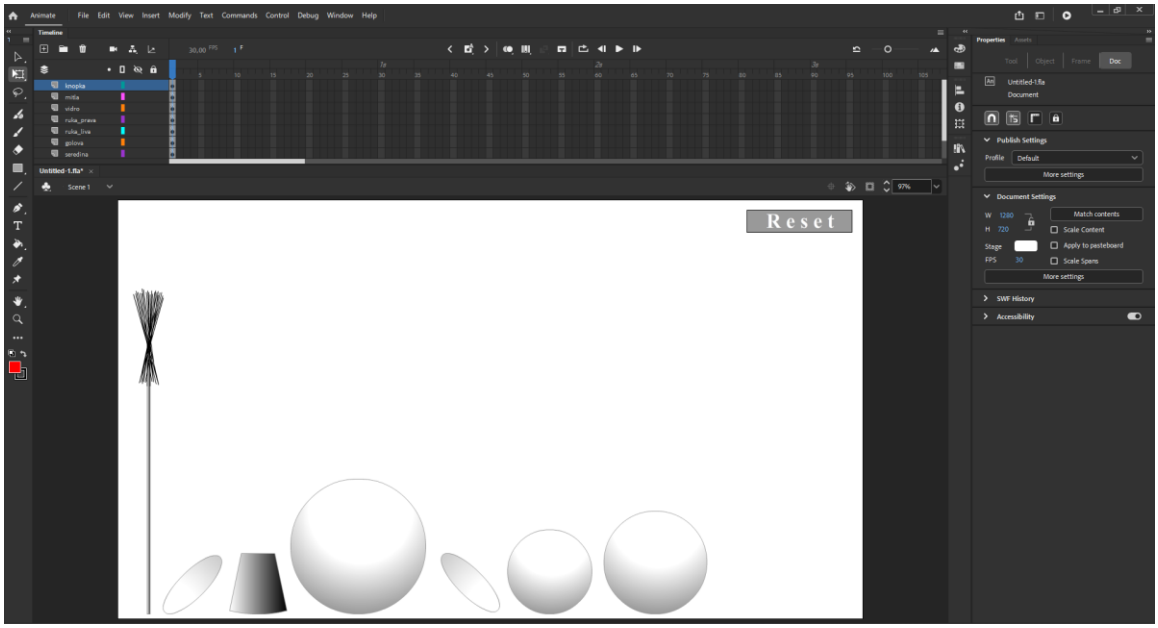
У кожному шарі під назвою кадрів ***Down*** клікаємо правою кнопкою миші по кадру і з контекстного меню вибираємо ***Insert Keyframe***. Отримаємо дублікати останніх зображень прямокутника і слова. За допомогою інструменту  ***Selection Tool*** виділяємо (обводимо) кнопку. За допомогою інструмента  ***Free Transform Tool*** і натиснутою клавішею ***Shift*** клавіатури змінюємо (зменшуємо вдвічі) розмір кнопки. Це створить ефект натискання на кнопку.

Розміщуємо кнопку на основній сцені.

Переходимо на основну сцену: натискаємо мишею на стрілочку ← зліва над білим полотном сцени.

Входимо у бібліотеку символів: натискаємо на іконку  , що ліворуч від панелі властивостей ***Properties***. З віконця бібліотеки (Library) перетаскуємо мишею зображення кнопки у верхню праву частину сцени.

Дайте назву кнопці. Для цього клікніть мишею по зображенню кнопки на сцені і у горі панелі властивостей ***Properties*** під параметром ***Button*** замість значення ***Instance Name*** напишіть ***Reset***. Ця назва буде використана при написанні коду для кнопки.



Зупиняємо відтворення анімації у першому кадрі: за допомогою інструменту  ***Selection Tool*** клікаємо у будь-якому шарі на ключовому кадрі, натискаємо клавішу ***F9*** і додаємо окремим рядком на початку коду команду: stop();

Додаємо пустий ключовий кадр на будь-якому шарі: клікаємо ***правою*** кнопкою миші на ***другому кадрі*** будь-якого шару і з контекстного меню вибираємо ***Insert Blank Keyframe***. Це дозволить очистити усі властивості об'єктів коли буде зчитаний другий кадр і заново повернуться властивості усіх об'єктів, коли буде зчитаний перший кадр.

Пишемо код для кнопки Reset.

Клікаємо по ***першому ключовому кадру*** шару ***knopka*** і натискаємо клавішу ***F9*** на клавіатурі. У вікні ***Action Script*** пишемо код:

```
Reset.addEventListener(MouseEvent.CLICK, onResetClick);
function onResetClick(e:MouseEvent):void {
   gotoAndPlay(2);
}
```

Протестуйте анімацію. Збережіть та опублікуйте документ.

Завдання для самостійної роботи:

Спробуйте самостійно додати нові елементи до анімації (морквину, шматочки вугілля, паличку) для відображення носу, ґудзиків та очей, рота сніговика. Вони також повинні мати властивість переміщуватися за допомогою миші.

Практична робота 5. Анімація з кількома сценами

***Завдання**: створити анімацію, яка складається з двох сцен: на першій сцені відбувається інтерактивна взаємодія вантажу і пружини (за допомогою миші треба підвісити вантаж на пружину), а друга сцена – відображення затухаючих коливань вантажу на пружині та побудова графіку коливань.*

Загальна логіка дій:

- ✓ *створити об'єкти, які будуть приймати участь в анімації (кожен об'єкт повинен бути на окремому шарі), перетворити об'єкти на символи, встановити необхідні властивості для об'єктів;*
- ✓ *написати код, який дозволяє переміщувати об'єкт при натиснутій на ньому кнопці миші й забороняє переміщування, коли кнопка миші відпущена;*
- ✓ *перевірити умову переходу на іншу сцену;*
- ✓ *у разі виконання умови реалізувати анімацію на іншій сцені;*
- ✓ *написати код для побудови графіку залежності змінної величини від часу.*

1. Створюємо об'єкти (вантаж і пружину)

Запускаємо програму ***Adobe Animate***.

У стартовому вікні вибираємо параметр ***Full HD***.

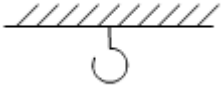
У верхньому правому куті сцени вибираємо значення масштабу ***Fit in Window***.

На даний момент існує один шар ***Layer_1***, ***індикатор відтворення*** знаходиться на ***першому пустому ключовому кадрі*** даного шару.

Почнемо з опори, до якої повинна бути прикріплена пружина.

Для зручності дамо назву шару ***fon***, через те, що на ньому будуть фонові об'єкти, які не прийматимуть безпосередньої участі в анімації. Двічі клікнемо лівою кнопкою миші по назві шару ***Layer_1,*** пишемо ***fon*** і натискаємо на клавіатурі клавішу ***Enter***.

За допомогою інструментів  ***Line Tool*** і  ***Oval Tool*** зображуємо у верхній лівій частині сцени підвіс, як показано на малюнку:

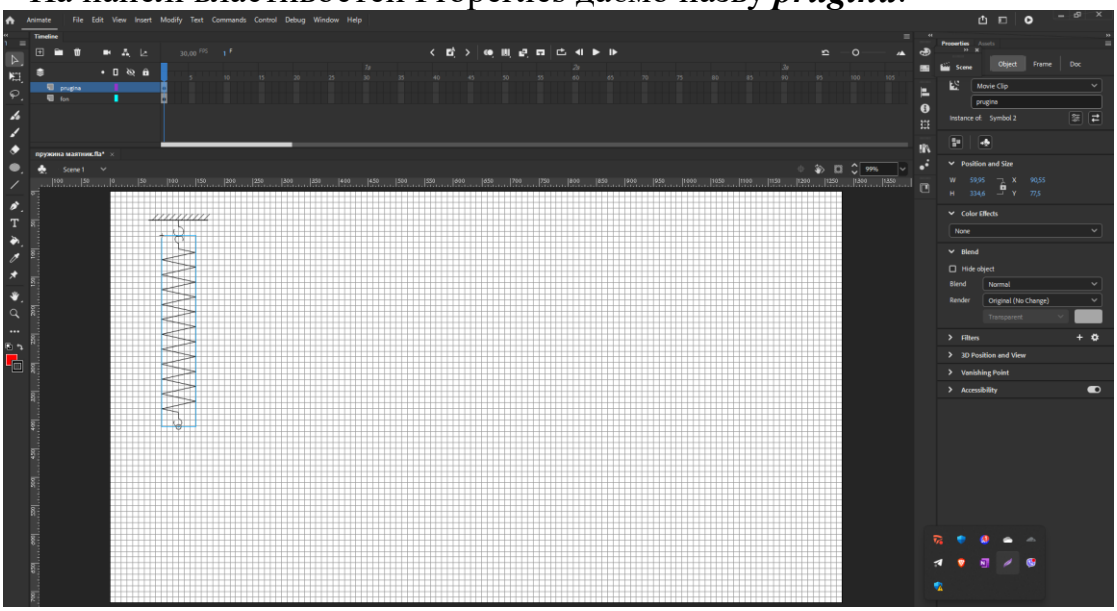


Створюємо ще один шар для зображення пружини, ***двічі клікнемо лівою кнопкою*** миші по назві шару ***Layer_2,*** пишемо ***prugina*** і натискаємо на клавіатурі клавішу ***Enter***.

Для зручності малювання пружини увімкнемо сітку, вибираємо на панелі меню ***View*** => ***Grid*** => ***Show Grid***.

За допомогою інструментів  ***Line Tool*** і  ***Oval Tool*** зображуємо пружину. Розташовуємо її так, нібито вона підчеплена до підвісу. Перетворюємо пружину на символ. Виділяємо пружину, клікнувши по шару ***prugina***, натискаємо клавішу ***F8*** на клавіатурі і у виниклому віконці встановлюємо значення параметру ***Type*** рівним ***Movie Clip***. Натискаємо ***OK***.

На панелі властивостей Properties даємо назву ***prugina***.

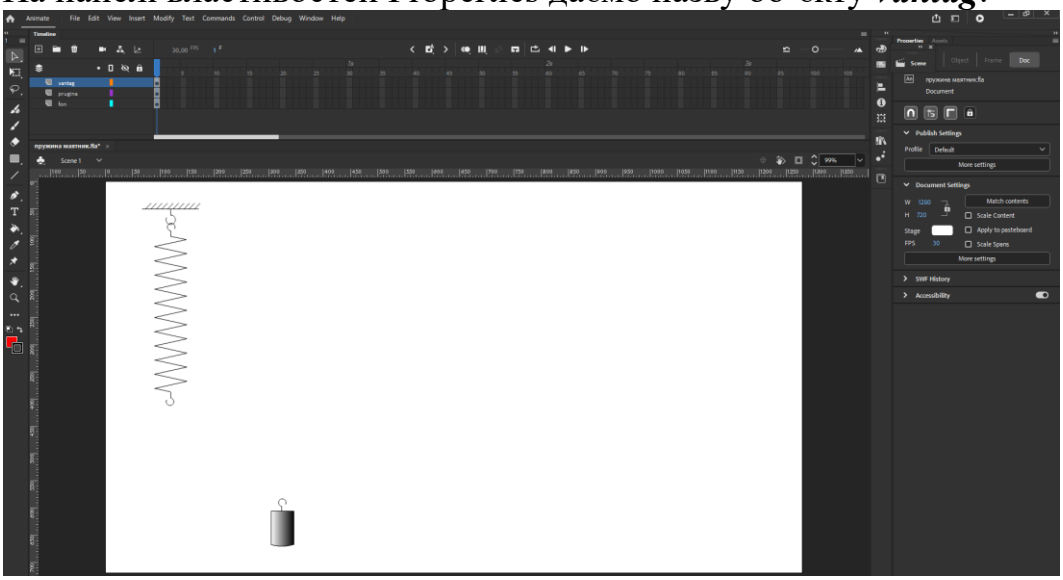


Додаємо ще один шар для зображення вантажу, який буде коливатися разом із пружиною. Клікаємо ***двічі лівою кнопкою*** миші по назві шару ***Layer_3,*** пишемо ***vantag*** і натискаємо на клавіатурі клавішу ***Enter***.

Якщо сітка заважає, її можна прибрати. Вибираємо на панелі меню ***View*** => ***Grid*** => ***Show Grid***.

За допомогою інструментів  Rectangle Tool,  ***Line Tool*** і  ***Oval Tool*** зображуємо вантаж. Розташовуємо його нижче і правіше від пружини. Перетворюємо вантаж на символ. Виділяємо вантаж, клікнувши по шару ***vantag***, натискаємо клавішу ***F8*** на клавіатурі і у виниклому віконці встановлюємо значення параметру ***Type*** рівним ***Movie Clip***. Натискаємо ***OK***.

На панелі властивостей Properties даємо назву об’єкту ***vantag***.



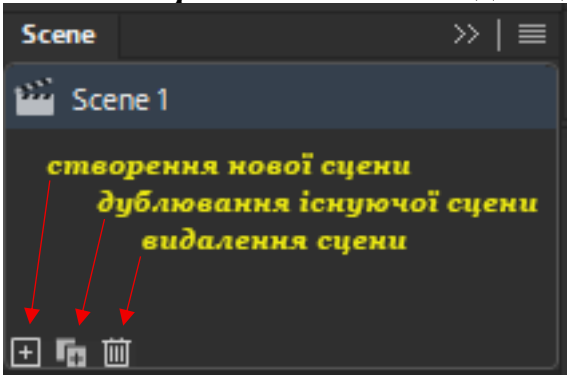
Напишемо код для об’єкта ***vantag*** у першому ключовому кадрі шару ***vantag***. У коді дозволимо переміщувати вантаж при натиснутій кнопці миші, а при відпусканні будемо порівнювати координати вантажу і пружини. Якщо координати збігатимуться в межах 10 пікселів, вантаж закріпеться на кінці пружини і перейде на сцену Scene 2, де ми створимо анімацію коливання вантажу на пружині і побудуємо графік незатухаючих коливань.

Натискаємо клавішу F9 на клавіатурі і пишемо код:

```
stop ();
// Додаємо слухачі подій для об'єкта vantag
vantag.addEventListener(MouseEvent.MOUSE_DOWN, startDragVantag);
stage.addEventListener(MouseEvent.MOUSE_UP, stopDragVantag);

function startDragVantag(e:MouseEvent):void {
   vantag.startDrag();
}

function stopDragVantag(e:MouseEvent):void {
   vantag.stopDrag();

   // Перевірка близькості координат vantag до prugina
   // Межа близькості 10 пікселів
   var dx:Number = Math.abs(vantag.x - prugina.x);
   var dy:Number = Math.abs((vantag.y - prugina.y)-320);
   var tolerance:Number = 20; // межа в пікселях

   if (dx <= tolerance && dy <= tolerance) {
      // Перехід на сцену Scene 2
      // Припускаємо, що сцена буде створена і названа саме так
      MovieClip(root).gotoAndPlay(1, "Scene 2");
   }
}
```

2. Створюємо сцену Scene 2 і на ній анімацію коливань

На панелі меню послідовно вибираємо ***Window*** => ***Scene***. У верхній частині панелі властивостей ***Properties*** з’явиться відповідне вікно:

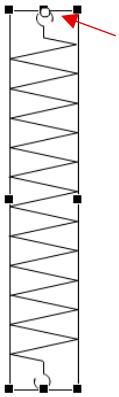


Продублюємо існуючу сцену ***Scene 1*** і назвемо її ***Scene 2***.

Після дублювання вміст обох сцен повністю ідентичний.

Підготуємо все для створення анімації коливань вантажу на пружині.

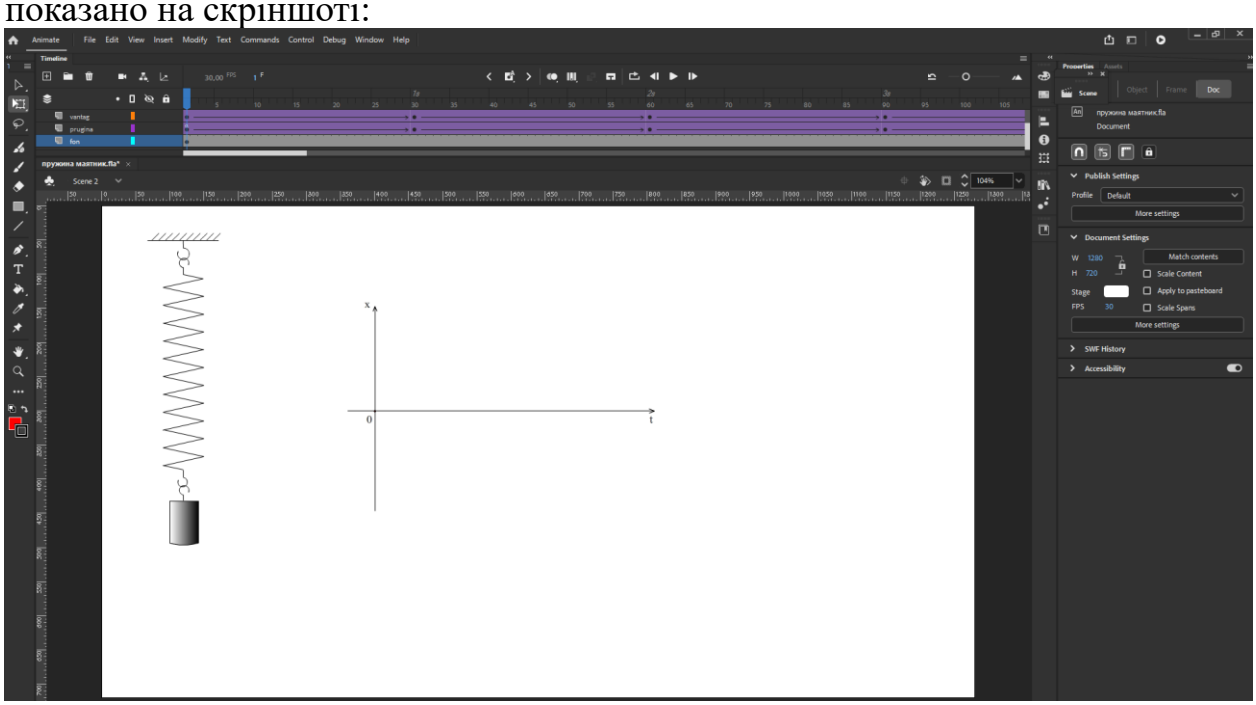
Переміщуємо вантаж (підчепимо його на пружину).



Встановимо точку відліку, відносно якої буде деформуватися пружина, на верхній край пружини. Для цього виділимо пружину (обведемо) за допомогою інструмента  Free Transform Tool. Посередині пружини побачимо білий кружечок, який і є точкою відліку. Перемістимо його вгору за допомогою миші.

Якщо цього не зробити, пружина буде деформуватися зверху і знизу одночасно відносно центру, а цього не повинно бути через те, що верхній край пружини закріплено на підвісі.

Клікнемо на першому (ключовому) кадрі шару ***fon*** і за допомогою інструментів  ***Line Tool*** і  ***Text Tool*** створимо систему координат, як показано на скріншоті:



Зробіть так, щоб початок координат знаходився в точці з координатами ***x = 400, y = 300***. Це можна зробити за допомогою зміни координат кожної осі на панелі властивостей ***Properties***, а також можна включити лінійки в меню, послідовно вибравши ***View*** => ***Rulers***.

Створюємо анімацію коливань, регулюючи розмір пружини і положення вантажу через кожні 30 кадрів на відповідних шарах.

Клікнемо ***правою кнопкою*** миші на ***30*** кадрі шару ***prugina*** і з контекстного меню виберемо ***Insert Keyframe***. За допомогою інструменту  ***Free Transform Tool*** розтягуємо пружину вниз. У ***першому*** (ключовому) кадрі шару клікаємо ***правою кнопкою*** миші і з контекстного меню вибираємо ***Create Classic Tween*** (створити класичну анімацію).

Клікнемо ***правою кнопкою*** миші на ***30*** кадрі шару ***vantag*** і з контекстного меню виберемо ***Insert Keyframe***. За допомогою миші або стрілок клавіатури

переміщуємо вантаж на нижній край пружини. У ***першому*** (ключовому) кадрі шару клікаємо ***правою кнопкою*** миші і з контекстного меню вибираємо ***Create Classic Tween*** (створити класичну анімацію).

Продовжуємо подібні дії на 60, 90 та 120 кадрах шарів ***prugina*** і ***vantag***, змінюючи відповідно розміри пружини і переміщуючи вантаж до нижнього кінця пружини.

Зациклимо анімацію, тим самим зробивши коливання незатухаючими. Для цього клікаємо у першому кадрі шару ***prugina***, натискаємо ***F9*** для визову вікна ***Action Script*** і пишемо код, яким повертаємо анімацію на 2 кадр, коли вона досягне 120 кадру:

```
// Слухаємо подію ENTER_FRAME — виконується кожен кадр
this.addEventListener(Event.ENTER_FRAME, checkFrame);

function checkFrame(e:Event):void {
   if (this.currentFrame == 120) {
      // Перехід на 2-й кадр цієї ж сцени
      this.gotoAndPlay(2);
     }
}
```

3. Створимо кнопку reset для повернення всієї анімації у початковий стан (Scene 1).

Створюємо ще один шар для зображення кнопки, двічі клікнемо ***лівою кнопкою*** миші по назві шару ***Layer_4,*** змінюємо назву шару на ***knopka*** і натискаємо на клавіатурі клавішу ***Enter***.

На верхній панелі меню вибираємо ***Insert => New Symbol***, у віконці, що з'явиться, встановлюємо у параметрі ***Type*** значення ***Button***.

За допомогою інструменту  ***Selection Tool*** вибираємо перший кадр кнопки (***Up***) шару ***Layer_1.***

За допомогою інструменту  ***Rectangle Tool*** з натиснутою клавішею Shift малюємо невеликий квадрат так, щоб його центр збігся із позначкою центру сцени (якщо не збігається – перемістіть за допомогою стрілок клавіатури). Зафарбуйте квадрат сірим кольором.

Додайте ще один шар ***Layer_2***. У першому кадрі цього шару поруч із квадратом за допомогою інструментів  ***Line Tool*** і  ***Oval Tool*** намалюйте заокруглену стрілку (вдвічі меншу за квадрат) і зафарбуйте її білим кольором. Перемістіть стрілку на центр квадрату (центр сцени).

Отримали вигляд кнопки «reset» у звичайному стані.

Змінимо вигляд кнопки, коли на неї наведена миша.

У кожному шарі під назвою кадрів ***Over*** клікаємо правою кнопкою миші по кадру і з контекстного меню вибираємо ***Insert Keyframe***. Отримаємо дублікати зображень квадрата і стрілки. Змінюємо їх колір заливки: квадрата на білий, а стрілки – на червоний.

Змінимо вигляд кнопки, коли на неї натиснули мишею.

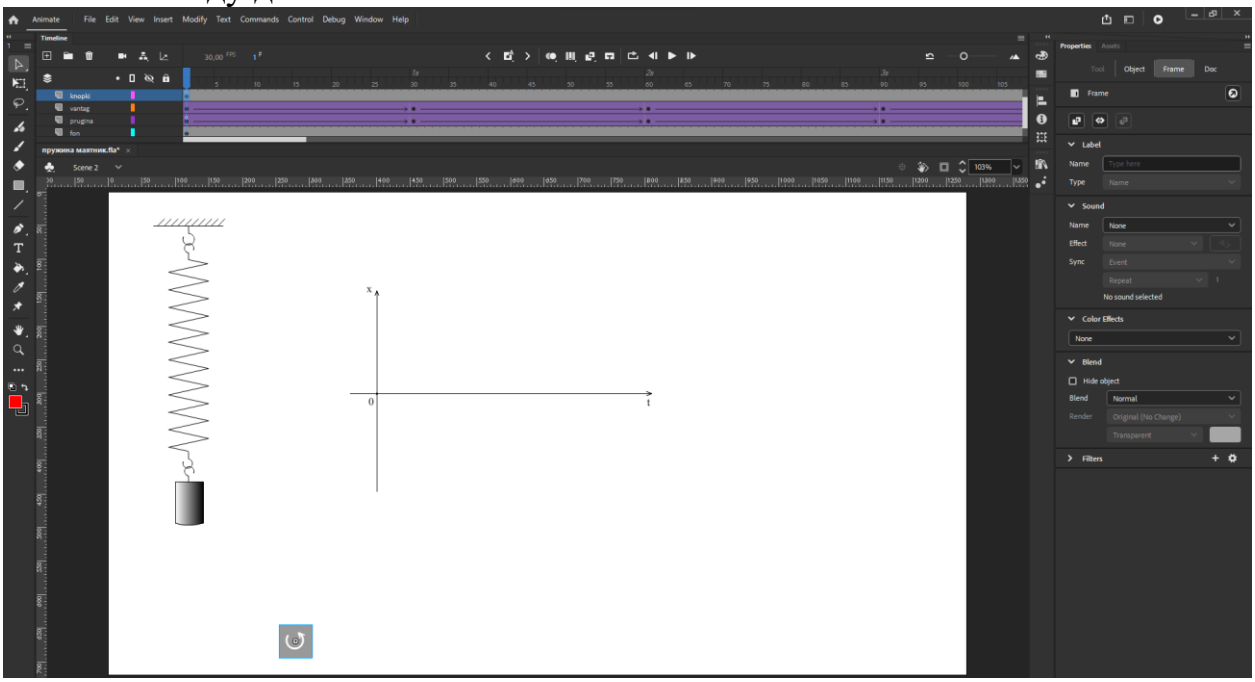
У кожному шарі під назвою кадрів ***Down*** клікаємо правою кнопкою миші по кадру і з контекстного меню вибираємо ***Insert Keyframe***. Отримаємо дублікати останніх зображень квадрата і трикутника. За допомогою інструменту  ***Free Transform Tool*** і натиснутою клавішею ***Shift*** клавіатури змінюємо (зменшуємо вдвічі) розмір кнопки. Це створить ефект натискання на кнопку.

Розміщуємо кнопку на основній сцені.

Переходимо на основну сцену: натискаємо мишею на стрілочку ← зліва над білим полотном сцени.

Входимо у бібліотеку символів: натискаємо на іконку  , що ліворуч від панелі властивостей ***Properties***. З віконця бібліотеки (Library) перетаскуємо мишею зображення кнопки нижче і праворуч від пружинного маятника.

Дайте назву кнопці. Для цього клікніть мишею по зображенню кнопки на сцені і у горі панелі властивостей ***Properties*** під параметром ***Button*** замість значення ***Instance Name*** напишіть ***reset***. Ця назва буде використана при написанні коду для кнопки.



Клікаємо по першому (ключовому) кадру шару ***knopka***, натискаємо на клавіатурі клавішу ***F9*** і у вікні ***Action Script*** пишемо код:

```
reset.addEventListener(MouseEvent.CLICK, onResetClick);

function onResetClick(event:MouseEvent):void {
   // Перехід на перший кадр сцени "Scene 1"
   MovieClip(root).gotoAndStop(1, "Scene 1");
}
```

4. Напишемо код для побудови графіку незатухаючих коливань.

Створюємо ще один шар для написання коду, двічі клікнемо ***лівою кнопкою*** миші по назві шару ***Layer_5,*** змінюємо назву шару на ***kod*** і натискаємо на клавіатурі клавішу ***Enter***.

Клікаємо по першому (ключовому) кадру шару ***kod***, натискаємо на клавіатурі клавішу ***F9*** і у вікні ***Action Script*** пишемо код:

```
import flash.display.Shape;
import flash.events.Event;
// Параметри графіка
var graph:Shape = new Shape();
addChild(graph);
var startX:Number = 400; // початок графіка по X
var startY:Number = 300; // початок графіка по Y
var frameCount:int = 0;
var maxFrames:int = 120;
var myscaleX:Number = 3; // масштаб по X (відстань між точками)
var amplitude:Number = 50; // амплітуда синусоїди
var frequency:Number = 0.1; // частота синусоїди
// Очищаємо графік і починаємо малювати лінію
graph.graphics.lineStyle(3, 0xFF0000);
graph.graphics.moveTo(startX, startY);
addEventListener(Event.ENTER_FRAME, onEnterFrame);
function onEnterFrame(e:Event):void {
   frameCount++;
      var xPos:Number = startX + frameCount * myscaleX;
      // Обчислюємо y за формулою синуса
   var yPos:Number = startY + amplitude * Math.sin(frequency * frameCount);
      graph.graphics.lineTo(xPos, yPos);
      if (frameCount >= maxFrames) {
      removeEventListener(Event.ENTER_FRAME, onEnterFrame);
   }
}
```

Протестуйте анімацію. Збережіть та опублікуйте документ.

Завдання для самостійної роботи:

Спробуйте самостійно додати до анімації пружинного маятника двонапрямлену стрілку, що відображала б довжину пружини під час коливань або проілюструйте видовження пружини під час коливань.

Практичне заняття 6. Створення анімації падіння тіла

***Завдання*: створити анімацію падіння тіла з урахуванням сили опору повітря, додати реалістичну анімацію відскоку після досягнення землі та реалізувати інтерфейс для плавної зміни коефіцієнта опору за допомогою слайдера.**

Падіння тіла з опором повітря описується рівнянням руху:

$$m \cdot \frac{dv}{dt} = m \cdot g - k \cdot v$$

У даному рівнянні:

m – маса тіла,

g – прискорення вільного падіння,

k – коефіцієнт опору повітря,

v – швидкість тіла.

Для моделювання використовуємо чисельний метод – на кожному кадрі будемо оновлювати швидкість і положення тіла.

Підготовка сцени

Запускаємо програму ***Adobe Animate***.

У стартовому вікні вибираємо параметр ***Full HD***.

У верхньому правому куті сцени вибираємо значення масштабу ***Fit in Window***.

Для створення анімації нам необхідно буде створити 6 шарів і розташувати на них відповідні об'єкти.

Створюємо перший шар для фону.

На даний момент існує один шар ***Layer_1***, ***індикатор відтворення*** знаходиться на ***першому пустому ключовому кадрі*** даного шару.

Двічі клікнемо лівою кнопкою миші по назві шару ***Layer_1,*** пишемо ***fon*** і натискаємо на клавіатурі клавішу ***Enter***.

Розмістимо на сцені картинку (небо і земля), яку можна завантажити з інтернету. Для цього у панелі меню послідовно натискаємо ***File => Import => Import to Stage***, у вікні, що з'явиться, вибираємо завантажену картинку і натискаємо відкрити. За допомогою інструменту  ***Free Transform Tool*** розтягніть картинку на всю сцену.

Створюємо другий шар для тіла, що падатиме

Додаємо ще один шар за допомогою інструмента  .

Змінюємо назву шару ***Layer_2*** на ***tilo***.

За допомогою інструменту  ***Oval Tool*** і з натиснутою на клавіатурі клавішею ***Shift*** (так буде малюватися коло) малюємо невелике коло.

Вибираємо інструмент  ***Selection Tool*** і двічі клікаємо мишею в середині кола – виділяємо його.

Змінюємо колір кола, натиснувши інструмент  ***Fill Color*** та вибравши один з кольорів (з центральним градієнтом) знизу палітри.

Клікнемо по назві шару ***tilo*** (виділимо коло). Далі вибираємо у верхньому меню ***Modify => Convert to Symbol*** або натискаємо на клавіатурі клавішу ***F8.*** З'явиться вікно перетворення об'єктів на символи. Встановлюємо параметр ***Movie Clip.***

На панелі властивостей ***Properties*** даємо назву об'єкту – під параметром ***Movie Clip*** замість значення ***Instance Name*** напишіть ***body_mc***. Ця назва буде використана при написанні коду.

Створюємо третій шар для виводу значення швидкості

Додаємо ще один шар за допомогою інструмента  .

Змінюємо назву шару ***Layer_3*** на ***speed***.

Вибираємо інструмент  ***Text Tool***, на панелі властивостей ***Properties*** замінюємо параметр ***Static Text*** (звичайний текст) на ***Dynamic Text*** (динамічний текст) і в лівій верхній частині сцени виділяємо широку і не високу область для тексту. Після цього на панелі властивостей ***Properties*** замінюємо ***Instance Name*** на ***velocity_txt,*** також встановлюємо колір тексту чорним, а розмір тексту ***Size*** рівним ***18***.

У створеному полі динамічного тексту буде відображатися значення поточної швидкості тіла.

Створюємо четвертий шар для кнопок

Додаємо ще один шар за допомогою інструмента  .

Змінюємо назву шару ***Layer_4*** на ***knopki***.

На цьому шарі створюємо дві кнопки – старт і стоп (дивись *Практичну роботу «Створення анімації руху заданою траєкторією з елементами інтерактивності»*).

Кнопки розміщуємо в лівій частині сцени.

Кнопці старт даємо назву ***startBtn***, а кнопці стоп – ***stopBtn*** на панелі властивостей ***Properties***, замінюючи для кнопок параметр ***Instance Name.***

Створюємо п'ятий шар для регулятора коефіцієнту опору і виведення його значення

Додаємо ще один шар за допомогою інструмента  .

Змінюємо назву шару ***Layer_5*** на ***slider***.

Вибираємо інструмент  ***Text Tool***, на панелі властивостей ***Properties*** замінюємо параметр ***Static Text*** на ***Dynamic Text*** і в лівій верхній частині сцени (між текстовим полем для швидкості і кнопками) виділяємо широку і не високу

область для тексту. Після цього на панелі властивостей ***Properties*** замінюємо ***Instance Name*** на ***kValue_txt,*** також встановлюємо колір тексту чорним, а розмір тексту ***Size*** рівним ***18***.

У створеному полі динамічного тексту буде відображатися поточне значення коефіцієнта опору повітря.

Додаємо на цей самий шар регулятор.

У панелі меню послідовно натискаємо ***Window => Components***, у вікні, що з'явиться, розкриваємо папку ***User Interface*** і переміщуємо мишею Slider на сцену під текстове поле для коефіцієнту опору. На панелі властивостей ***Properties*** замінюємо ***Instance Name*** на ***kSlider***.

Створеним слайдером будемо перемикати значення опору повітря.

Створюємо шостий шар для написання коду

Додаємо ще один шар за допомогою інструмента  .

Змінюємо назву шару ***Layer_6*** на ***kod***.

Даний шар залишаємо пустим. У першому кадрі буде розміщений весь код ActionScript 3.0.

Після всіх виконаних нами дій отримуємо такий вигляд сцени:

Написання коду

Клікаємо по першому (пустому ключовому) кадру шару з іменем ***kod*** і відкриваємо панель **ActionScript,** натиснувши клавішу F9 на клавіатурі.

У вікні, що з'явиться, пишемо наступний повний код:

```
import fl.events.SliderEvent;

// Фізичні константи та змінні
var g:Number = 9.8;              // Прискорення вільного падіння (пікселі/с²)
var mass:Number = 1;             // Маса тіла (умовна одиниця)
var k:Number = 0.1;              // Початковий коефіцієнт опору повітря

var velocity:Number = 0;         // Початкова швидкість (пікселі/кадр)
var dt:Number = 1 / stage.frameRate;  // Крок часу (секунди на кадр)
var isFalling:Boolean = false;   // Статус анімації

// Параметри відскоку
var restitution:Number = 0.7;    // Коефіцієнт відскоку (0-1)

// Початкове положення тіла
body_mc.x = stage.stageWidth / 2;
body_mc.y = 50;

// Оновлення текстового поля швидкості
function updateVelocityText():void {
    velocity_txt.text = "Швидкість: " + velocity.toFixed(2) + " пікс/кадр";
}

// Оновлення тексту коефіцієнта опору
function updateKValueText():void {
    kValue_txt.text = "Опір повітря (k): " + k.toFixed(2);
}

// Функція оновлення руху на кожному кадрі
function updateFall(e:Event):void {
    if (!isFalling) return;

    var Fg:Number = mass * g;          // Сила тяжіння
    var Fr:Number = -k * velocity;     // Сила опору

    var F:Number = Fg + Fr;            // Загальна сила
    var a:Number = F / mass;           // Прискорення

    velocity += a * dt;                // Оновлення швидкості
    body_mc.y += velocity;             // Оновлення позиції

    updateVelocityText();

    // Відскок при досягненні "землі"
    var groundY:Number = stage.stageHeight - body_mc.height / 2;
```

```
        if (body_mc.y > groundY) {
            body_mc.y = groundY;
            velocity = -velocity * restitution; // Зворотній рух зі зменшеною швидкістю

            // Якщо швидкість дуже мала — зупинити анімацію
            if (Math.abs(velocity) < 0.5) {
                isFalling = false;
                removeEventListener(Event.ENTER_FRAME, updateFall);
                trace("Падіння завершено");
            }
        }
    }

    // Запуск анімації
    function startFall(e:MouseEvent):void {
        velocity = 0;
        body_mc.y = 50;
        isFalling = true;
        addEventListener(Event.ENTER_FRAME, updateFall);
        updateVelocityText();
    }

    // Зупинка анімації
    function stopFall(e:MouseEvent):void {
        isFalling = false;
        removeEventListener(Event.ENTER_FRAME, updateFall);
    }

    // Обробник зміни слайдера
    function onSliderChange(e:SliderEvent):void {
        k = e.value;
        updateKValueText();
    }

    // Прив'язка кнопок та слайдера до функцій
    startBtn.addEventListener(MouseEvent.CLICK, startFall);
    stopBtn.addEventListener(MouseEvent.CLICK, stopFall);
    kSlider.addEventListener(SliderEvent.CHANGE, onSliderChange);

    // Ініціалізація тексту і слайдера
    kSlider.minimum = 0;
    kSlider.maximum = 1;
    kSlider.value = k;
    updateKValueText();
    updateVelocityText();
```

Пояснення коду

- Константи **g**, **mass**, **k** задають фізичні параметри: прискорення вільного падіння, масу тіла і коефіцієнт опору повітря.
- **velocity** – поточна швидкість тіла.
- **dt** – крок часу, розрахований на основі частоти кадрів сцени.
- **isFalling** – прапорець, що контролює, чи активна анімація.
- **restitution** – коефіцієнт відскоку (0 – без відскоку, 1 – ідеальний відскок).
- Функція **updateFall** виконується на кожному кадрі, обчислює сили, прискорення, оновлює швидкість і положення тіла.
- При досягненні "землі" тіло відскакує, швидкість змінює знак і зменшується.
- Якщо швидкість стає дуже малою, анімація зупиняється.
- Функції **startFall** і **stopFall** керують запуском і зупинкою анімації.
- Слайдер **kSlider** дозволяє змінювати коефіцієнт опору, а текстове поле **kValue_txt** відображає його значення.

Тестування

- Запустіть анімацію (Ctrl + Enter).
- Переміщуйте повзунок ***слайдера***, щоб змінити опір повітря.
- Натисніть кнопку ***старт*** – тіло почне падати, швидкість відображатиметься у текстовому полі.
- Після досягнення землі тіло відскочить із амплітудою, що залежить від коефіцієнта відскоку.
- Натисніть кнопку ***стоп*** для зупинки анімації.

Ця розширена модель падіння тіла з урахуванням опору повітря, анімацією відскоку та інтерактивним регулятором дає змогу не лише візуалізувати фізичний процес, а й експериментувати з параметрами в реальному часі. Такий підхід сприяє глибшому розумінню фізичних законів і розвитку дослідницьких навичок.

Побудова ліній засобами ActionScript 3.0

ActionScript 3.0 надає потужний та гнучкий інструментарій для динамічного створення графіки, зокрема малювання ліній і фігур безпосередньо з коду. Для цього задля промальовування ліній використовують властивість ***graphics*** об'єктів типу ***Shape***, ***Sprite*** або ***MovieClip***.

Це важливо для створення кастомних анімацій, візуалізацій, інструментів або ігор, де графічна частина динамічно управляється кодом.

Основні поняття

- **graphics** – властивість об'єкта, яка містить об'єкт типу ***graphics***. Саме він відповідає за малювання графіки у внутрішньому контексті об'єкта.
- **lineStyle(thickness, color, alpha)** – задає стиль лінії: товщину у пікселях, колір (16-річний код), прозорість (0 - 1).
- **moveTo(x, y)** – встановлює поточну позицію пера в точку (x, y) на площині.
- **lineTo(x, y)** – малює лінію від поточної позиції пера до точки (x, y) і переносить «перо» в цю точку.

Покрокове створення лінії у ActionScript 3.0

Створення об'єкта для малювання

```
var myShape:Shape = new Shape();
addChild(myShape);  // Додаємо лінію на сцену
```

Пояснення: створили порожній графічний об'єкт типу ***Shape***, додали його на сцену, щоб він відобразився.

Встановлення стилю лінії

```
myShape.graphics.lineStyle(2, 0xFF0000, 1);
```

Пояснення: товщина лінії 2 пікселі, колір – червоний (0xFF0000), прозорість – 1 (повністю непрозора).

Вказівка точки початку малювання

```
myShape.graphics.moveTo(50, 50);
```

Починаємо малювати з координат (50, 50).

Малювання лінії

```
myShape.graphics.lineTo(200, 200);
```

Малюємо лінію від (50, 50) до (200, 200).

Весь код разом

```
var myShape:Shape = new Shape();
addChild(myShape);
myShape.graphics.lineStyle(2, 0xFF0000, 1);
myShape.graphics.moveTo(50, 50);
myShape.graphics.lineTo(200, 200);
```

Малювання фігур лініями

Для побудови ломаних ліній можна послідовно виконувати команду ***lineTo()*** до потрібних точок.

Для замкнених фігур можна використовувати метод ***graphics.lineTo()***

останньою точкою, щоб повернутися в початок. Для заливки фігури використовується ***graphics.beginFill()*** і ***graphics.endFill()***.

Треба враховувати важливі моменти:

- **лінія завжди малюється від останньої позиції пера.** Спочатку виконується ***moveTo()*** для встановлення початку, і далі ***lineTo()*** для кожного відрізку;
- **стиль лінії застосовується до всіх наступних малюнків.** Кожний новий lineStyle() перезаписує попередній;
- **якщо не викликати lineStyle(), лінії будуть невидимими.**
- Малювання відбувається відносно локальної системи координат об'єкта.

Приклад: малювання прямокутника з ліній

```
var rectShape:Shape = new Shape();
addChild(rectShape);

rectShape.graphics.lineStyle(3, 0x0000FF, 1);
rectShape.graphics.moveTo(100, 100);
rectShape.graphics.lineTo(300, 100);
rectShape.graphics.lineTo(300, 200);
rectShape.graphics.lineTo(100, 200);
rectShape.graphics.lineTo(100, 100);
```

Додаткові методи

- ***graphics.clear()*** – очищення усієї графіки в об'єкті.
- ***graphics.beginFill(color, alpha)*** – початок заливки фігури.
- ***graphics.endFill()*** – закінчення заливки.
- ***graphics.lineGradientStyle()*** – створення ліній з градієнтом.

Зауваження

- Для динамічної анімації ліній створюються об'єкти ***Shape*** або ***Sprite*** і оновлюється їх графіка в події ***Event.ENTER_FRAME***.
- Якщо потрібно змінити малюнок – треба виконати команду ***graphics.clear()*** перед новим малюванням.

Побудова суцільної лінії заданого стилю

Щоб намалювати лінію певного стилю з одних координат в інші потрібно:

1. Використати об'єкт ***graphics*** (наприклад, у ***Shape*** або на головному об'єкті сцени).
2. Задати стиль лінії за допомогою ***lineStyle(thickness, color, alpha)***, де:
   - ***thickness*** – товщина лінії (у пікселях),
   - ***color*** – колір у форматі 0xRRGGBB,
   - ***alpha*** – прозорість від 0 до 1 (1 – повна непрозорість).
3. Встановити початкову точку лінії за допомогою ***moveTo(x, y)***.
4. Провести лінію до кінцевої точки за допомогою ***lineTo(x, y)***.

Приклад:

```
// створюємо Shape для малювання
var lineShape:Shape = new Shape();
addChild(lineShape);

// задаємо товщину 3 пікселя, колір синій, повна непрозорість
lineShape.graphics.lineStyle(3, 0x0000FF, 1);

// початкова точка лінії
lineShape.graphics.moveTo(startX, startY);

// кінцева точка лінії
lineShape.graphics.lineTo(endX, endY);
```

Побудова пунктирної лінії заданого стилю

Щоб намалювати пунктирну лінію певного стилю між двома координатами, потрібно малювати не одну суцільну лінію, а розбити її на багато коротких штрихів із проміжками.

Кроки для малювання пунктирної лінії:

1. Створіть об'єкт для малювання – зазвичай ***Shape***.
2. Очищуйте графіку перед малюванням, щоб не накопичувались попередні лінії.
3. Встановіть стиль лінії через ***lineStyle(товщина, колір, прозорість)***.
4. Розрахуйте відстань і кут між початковою і кінцевою точками.
5. Циклічно малюйте маленькі штрихи (***dash***) і пропуски (***gap***) за допомогою тригонометрії:
   - Починаємо з початкової точки.
   - Малюємо штрих завдовжки ***dashLength***.
   - Пропускаємо проміжок ***gapLength***.
   - Переходимо до наступного штриха, поки не досягнемо кінця.

Приклад коду пунктирної лінії товщиною 3 пікселі, синього кольору між точками (startX, startY) та (endX, endY):

```
import flash.display.Shape;

var lineShape:Shape = new Shape();
addChild(lineShape);

var dashLength:Number = 10;  // довжина штриха
var gapLength:Number = 5;    // довжина проміжку

function drawDashedLine(startX:Number, startY:Number, endX:Number,
endY:Number):void {
   lineShape.graphics.clear();
   lineShape.graphics.lineStyle(3, 0x0000FF); // товщина 3, синій колір
```

```
    var dx:Number = endX - startX;
    var dy:Number = endY - startY;
    var lineLength:Number = Math.sqrt(dx * dx + dy * dy);
    var angle:Number = Math.atan2(dy, dx);

    var currentLength:Number = 0;
    var x:Number = startX;
    var y:Number = startY;

    while (currentLength < lineLength) {
        var dashEndLength:Number = Math.min(dashLength, lineLength -
currentLength);

        var dashEndX:Number = x + Math.cos(angle) * dashEndLength;
        var dashEndY:Number = y + Math.sin(angle) * dashEndLength;

        lineShape.graphics.moveTo(x, y);
        lineShape.graphics.lineTo(dashEndX, dashEndY);

        // Переходимо через проміжок gapLength
        currentLength += dashLength + gapLength;
        x = x + Math.cos(angle) * (dashLength + gapLength);
        y = y + Math.sin(angle) * (dashLength + gapLength);
    }
}

// Приклад виклику: лінія між (50, 50) і (200, 150)
drawDashedLine(50, 50, 200, 150);
```

Динамічна побудова лінії заданого стилю між двома рухомими об’єктами

Наприклад, на сцені зображено два об’єкти (кульки) типу ***movieClip*** на окремих шарах. Об’єкти мають імена ***instance name*** redkulka та greenkulka.

Напишемо код, який дозволить перетягувати два об'єкти (redkulka та greenkulka) і при цьому малювати синю лінію товщиною 3 пікселі, що з'єднує ці об'єкти.

Код слід розмістити на головному кадрі сцени.

Код для випадку суцільної лінії:

```
import flash.events.MouseEvent;
import flash.display.Shape;
import flash.display.Sprite;
import flash.geom.Point;

// Shape для малювання лінії
var connectionLine:Shape = new Shape();
```

```
addChild(connectionLine);

// Змінна для відстеження об'єкта, що перетягується
var draggedObj:Sprite = null;

// Початок перетягування об'єкта
function startDragObj(e:MouseEvent):void {
    draggedObj = e.currentTarget as Sprite;
    draggedObj.startDrag();
    stage.addEventListener(MouseEvent.MOUSE_MOVE, onMouseMove);
    stage.addEventListener(MouseEvent.MOUSE_UP, stopDragObj);
    drawLine();
}

// Завершення перетягування
function stopDragObj(e:MouseEvent):void {
    if (draggedObj) {
        draggedObj.stopDrag();
        draggedObj = null;
        stage.removeEventListener(MouseEvent.MOUSE_MOVE, onMouseMove);
        stage.removeEventListener(MouseEvent.MOUSE_UP, stopDragObj);
        drawLine(); // Оновити лінію
    }
}

// Малювання суцільної синьої лінії товщиною 3 пікселі між центрами об'єктів
function drawLine():void {
    connectionLine.graphics.clear();
    connectionLine.graphics.lineStyle(3, 0x0000FF); // товщина 3, синій колір

    // Центри об'єктів (підлаштовано під їхні width (ширину) і height (висоту))
    var pt1:Point = new Point(redkulka.x + redkulka.width / 2, redkulka.y +
redkulka.height / 2);
    var pt2:Point = new Point(greenkulka.x + greenkulka.width / 2, greenkulka.y +
greenkulka.height / 2);
    connectionLine.graphics.moveTo(pt1.x, pt1.y);
    connectionLine.graphics.lineTo(pt2.x, pt2.y);
}

// Прив’язуємо події перетягування до об’єктів
redkulka.addEventListener(MouseEvent.MOUSE_DOWN, startDragObj);
greenkulka.addEventListener(MouseEvent.MOUSE_DOWN, startDragObj);

// Від початкового положення малюємо лінію
drawLine();
```

Код для випадку пунктирної лінії:

```
import flash.events.MouseEvent;
import flash.display.Shape;
import flash.display.Sprite;
import flash.geom.Point;

// Shape для малювання лінії
var connectionLine:Shape = new Shape();
addChild(connectionLine);

// Для відстеження вибраного об'єкта
var draggedObj:Sprite = null;

// Обробник початку перетягування
function startDragObj(e:MouseEvent):void {
   draggedObj = e.currentTarget as Sprite;
   draggedObj.startDrag();
   stage.addEventListener(MouseEvent.MOUSE_MOVE, onMouseMove);
   stage.addEventListener(MouseEvent.MOUSE_UP, stopDragObj);
   drawLine();
}

// Обробник зупинки перетягування
function stopDragObj(e:MouseEvent):void {
   if(draggedObj){
      draggedObj.stopDrag();
      draggedObj = null;
      stage.removeEventListener(MouseEvent.MOUSE_MOVE, onMouseMove);
      stage.removeEventListener(MouseEvent.MOUSE_UP, stopDragObj);
      drawLine();
   }
}

// Малювання пунктирної синьої лінії товщиною 3
function drawLine():void {
   connectionLine.graphics.clear();
   var pt1:Point = new Point(redkulka.x + redkulka.width/2, redkulka.y +
redkulka.height/2);
   var pt2:Point = new Point(greenkulka.x + greenkulka.width/2, greenkulka.y +
greenkulka.height/2);

   connectionLine.graphics.lineStyle(3, 0x0000FF); // Синій колір, товщина 3

   // Малюємо пунктирну лінію:
   var dashLength:Number = 10;
```

```
    var gapLength:Number = 5;
    var totalLength:Number = Point.distance(pt1, pt2);
    var numDashes:int = totalLength / (dashLength + gapLength);
    var angle:Number = Math.atan2(pt2.y - pt1.y, pt2.x - pt1.x);

    var startX:Number = pt1.x;
    var startY:Number = pt1.y;

    for(var i:int = 0; i < numDashes; i++){
        var endX:Number = startX + Math.cos(angle) * dashLength;
        var endY:Number = startY + Math.sin(angle) * dashLength;
        connectionLine.graphics.moveTo(startX, startY);
        connectionLine.graphics.lineTo(endX, endY);
        startX = endX + Math.cos(angle) * gapLength;
        startY = endY + Math.sin(angle) * gapLength;
    }
}

// Оновлення лінії під час руху об'єкта
function onMouseMove(e:MouseEvent):void {
    drawLine();
}

// Прив'язуємо обробники до об'єктів
redkulka.addEventListener(MouseEvent.MOUSE_DOWN, startDragObj);
greenkulka.addEventListener(MouseEvent.MOUSE_DOWN, startDragObj);

// Від початкового положення малюємо пунктирну лінію
drawLine();
```

Побудова графіків тригонометричних функцій із заданими параметрами

Щоб намалювати синусоїду або косинусоїду певного стилю, починаючи із заданої точки, потрібно програмно малювати послідовність точок, які описують графік функції синуса або косинуса, та з'єднувати їх лініями.

Основні параметри, які дозволять задати форму графіка:

- Початкова точка – координати (startX, startY), з якої починаємо малювати графік.
- Довжина графіка – скільки пікселів по горизонталі ми хочемо побудувати.
- Висота (амплітуда) – максимальне відхилення графіка по вертикалі.
- Частота чи довжина хвилі – визначає, скільки коливань буде на довжині графіка (або довжина одного періоду в пікселях).

- Стиль лінії – товщина, колір, пунктирність (визначається через ***lineStyle*** і логіку малювання).

Нижче наведено приклад коду, який виводить синусоїдальний графік:

```
import flash.display.Shape;

// Створюємо Shape для малювання
var sineShape:Shape = new Shape();
addChild(sineShape);

function drawSineWave(startX:Number, startY:Number, length:Number,
amplitude:Number, wavelength:Number, thickness:Number, color:uint):void {
   sineShape.graphics.clear();
   sineShape.graphics.lineStyle(thickness, color);

   var points:int = length; // кількість точок (пикселей по горизонталі)

   sineShape.graphics.moveTo(startX, startY);

   for (var i:int = 0; i <= points; i++) {
      // Обчислюємо координату X по горизонталі
      var xPos:Number = startX + i;

      // Обчислюємо координату Y по формулі синуса
      // Частота синуса = 2π / wavelength
      var yPos:Number = startY + Math.sin((2 * Math.PI / wavelength) * i) *
amplitude;

      sineShape.graphics.lineTo(xPos, yPos);
   }
}

// Приклад виклику - малюємо синусоїду
// Починаємо з точки (50, 100), довжина 300px, висота (амплітуда) 50px,
// довжина хвилі (період) 100px, товщина лінії 3, колір синій (0x0000FF)
drawSineWave(50, 100, 300, 50, 100, 3, 0x0000FF);
```

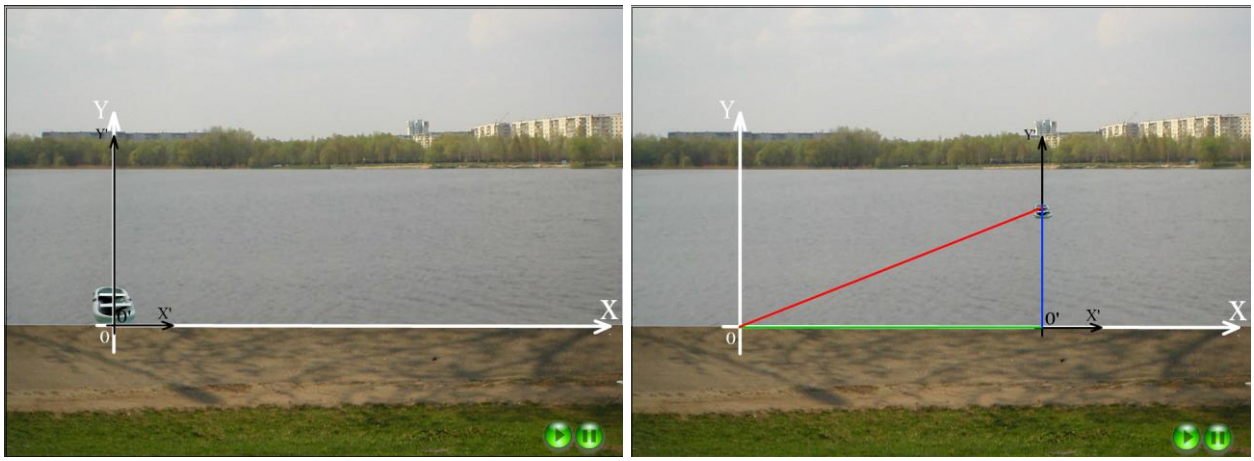
Для відображення косинусоїди, треба змінити команду ***Math.sin*** на ***Math.cos*** у формулі ***yPos***.

Завдання для самостійної роботи

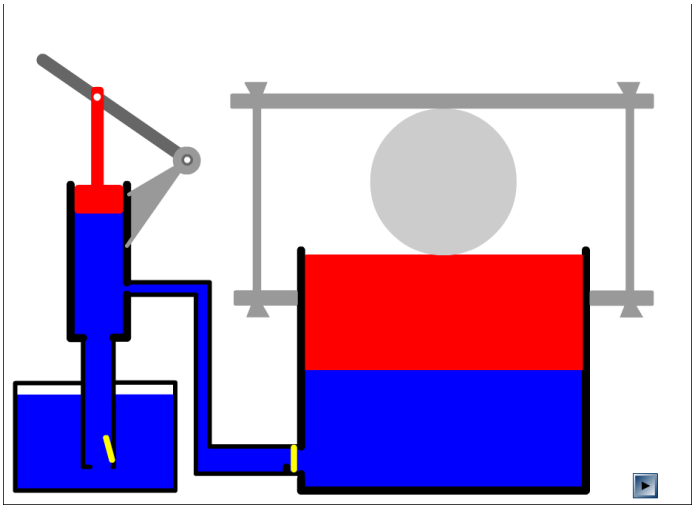
<u>Завдання:</u> користуючись графічним редактором Adobe Animate створити інтерактивну анімацію фізичного досліду / явища / процесу, що дозволить з'ясувати певні істотні ознаки вибраного вами фізичного поняття (величини, закону, теорії, явища, технічного пристрою або приладу).

Теми власних проєктів:

1. **Класичний закон додавання швидкостей (переміщень)**
   Анімація відносного руху (наприклад, човен перпендикулярно течії перепливає річку) з різними швидкостями, з візуальним показом сумарної швидкості (переміщення).
   Приблизний вигляд сцени:



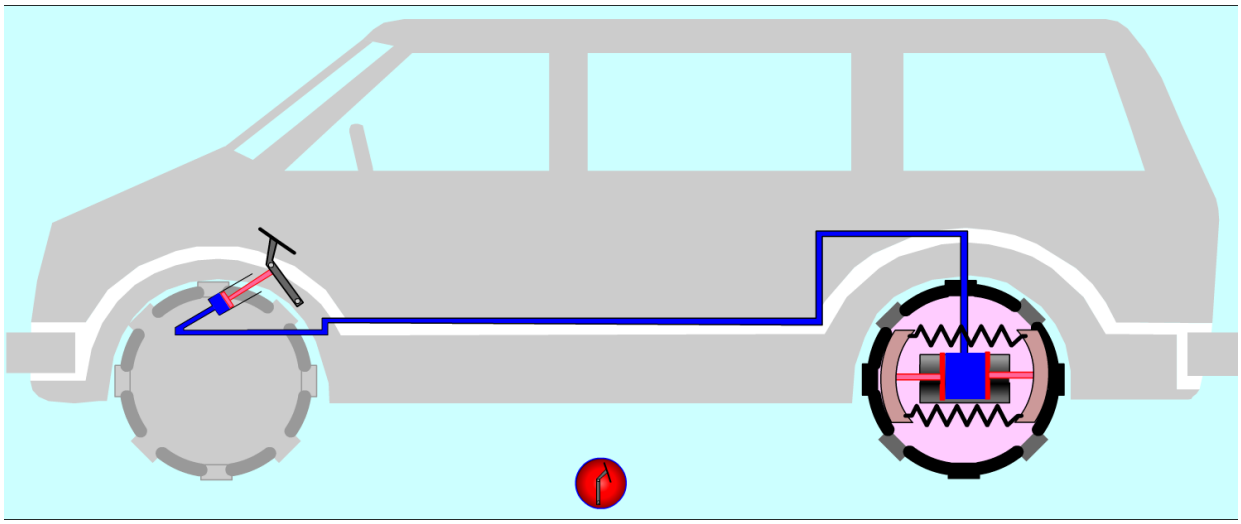
2. **Гідравлічний прес**
   Демонстрація принципу роботи преса: взаємодія площ поршнів і передача сили з підсиленням.
   Приблизний вигляд сцени:



3. **Гідравлічне гальмо**

Анімація роботи гальмівної системи з передачею тиску через рідину на гальмівні колодки.

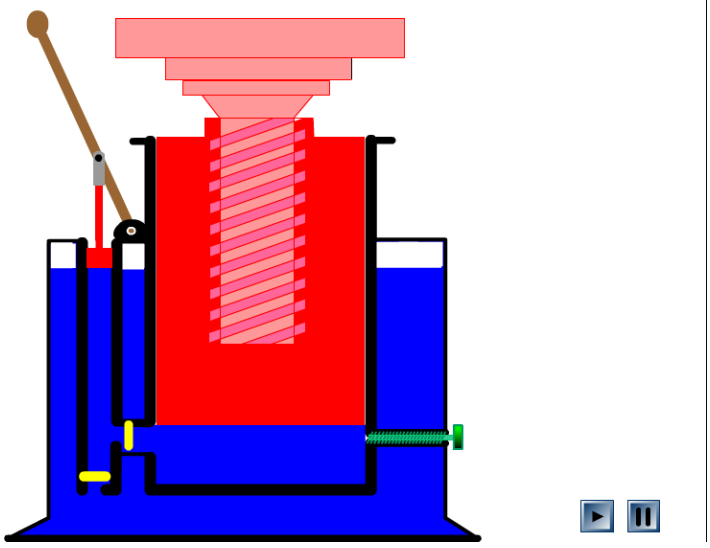
Приблизний вигляд сцени:



4. **Гідравлічний домкрат**

Візуалізація підйому вантажу завдяки дії гідравлічного тиску на різні площі поршнів.

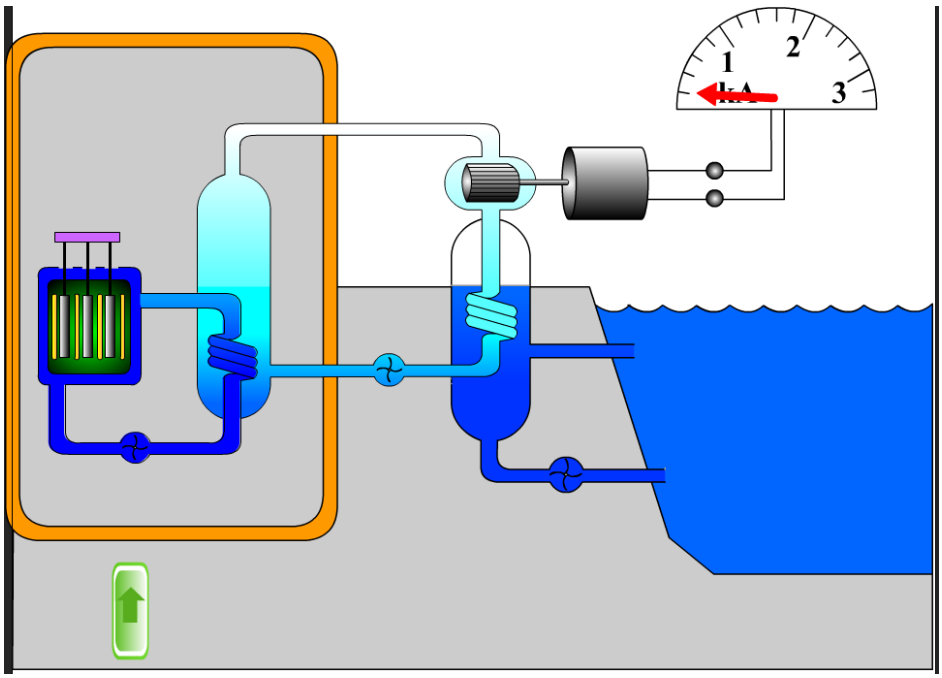
Приблизний вигляд сцени:



5. **Ядерний реактор**

Показ процесу нагрівання води, утворення пари, конденсації, контролю потужності, роботи генератора.

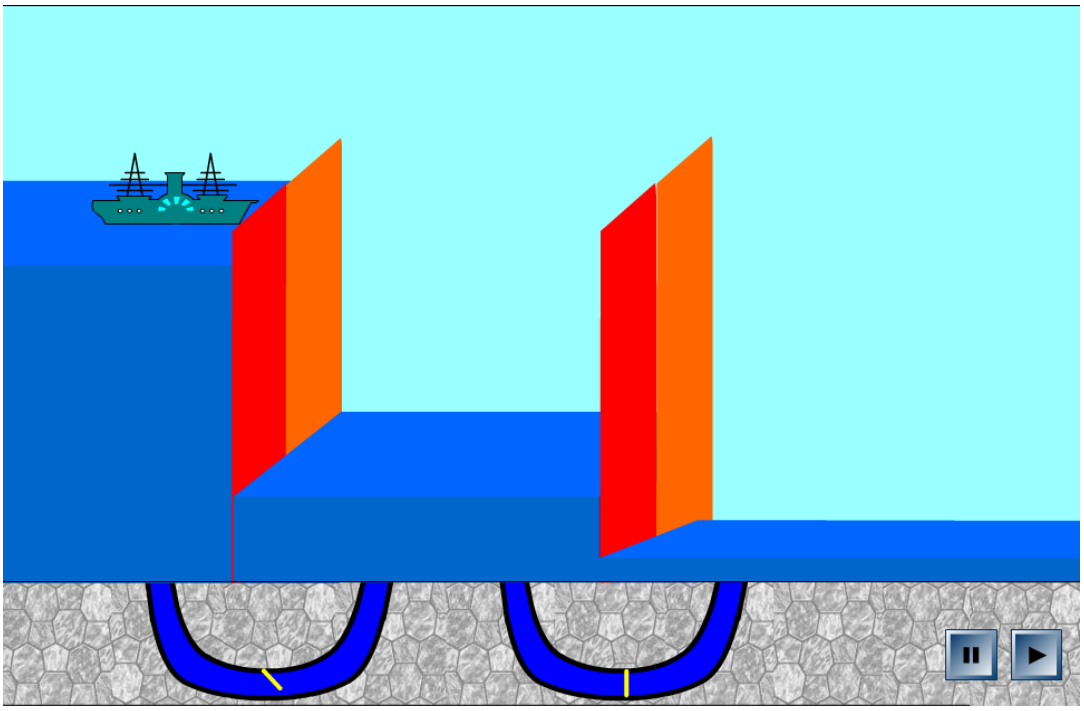
Приблизний вигляд сцени:



6. **Шлюзи**

Анімація роботи шлюзів: наповнення та випуск води для підняття/опускання суден (робота клапанів, принцип сполучених посудин).

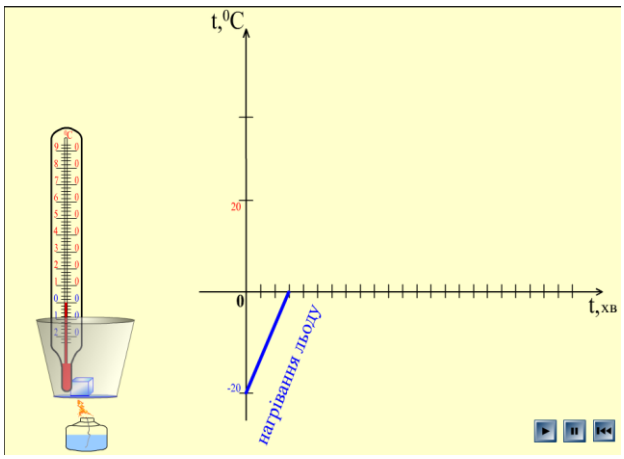
Приблизний вигляд сцени:

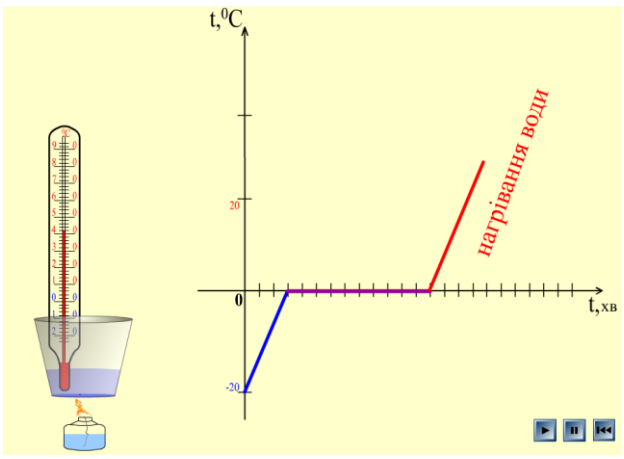


7. **Графік танення льоду**

Інтерактивний графік залежності температури від часу під час танення льоду з анімацією зміни стану.

Приблизний вигляд сцени:

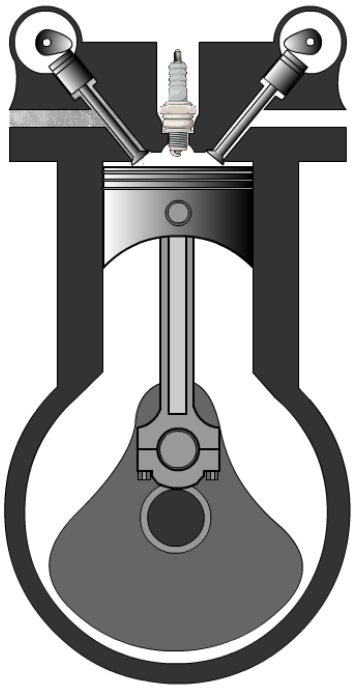




8. Двигун внутрішнього згоряння

Показ циклу роботи двигуна: впуск, стиснення, згоряння, випуск.

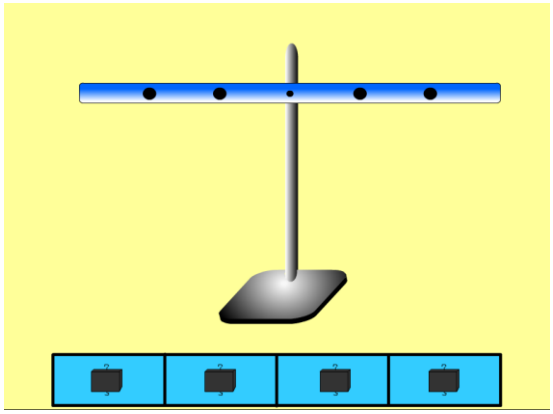
Приблизний вигляд сцени:



9. Важіль: правило моментів

Анімація з важелем, де можна змінювати довжини плечей і сили, щоб побачити умову рівноваги.

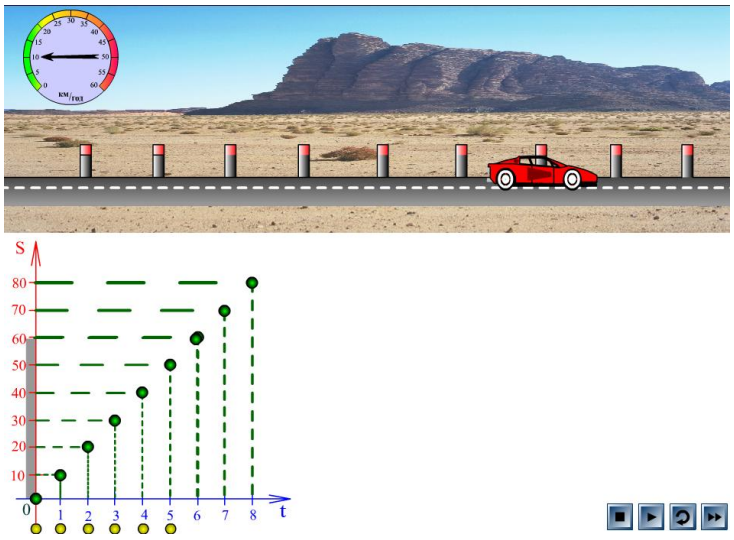
Приблизний вигляд сцени:



10. Графік зміни швидкості тіла (рівномірний рух)

Побудова графіка швидкості з анімацією руху тіла з постійною швидкістю.

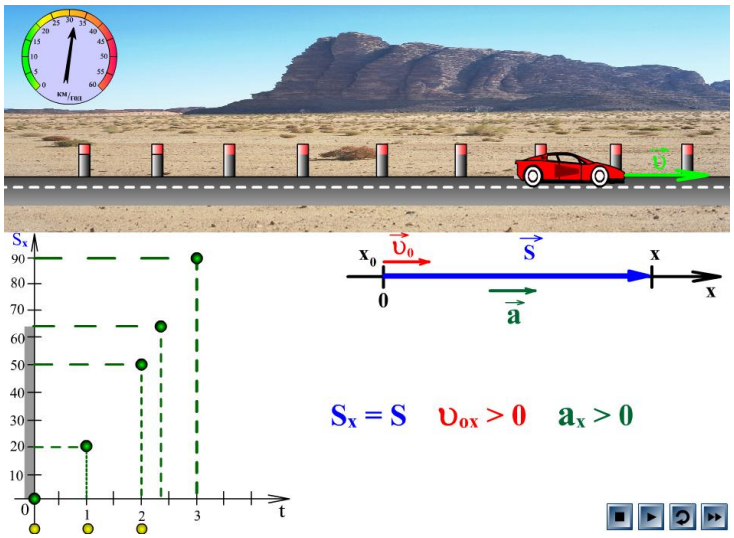
Приблизний вигляд сцени:



11. Графік зміни швидкості тіла (прискорений рух)

Анімація руху тіла з прискоренням і відповідний графік зміни швидкості.

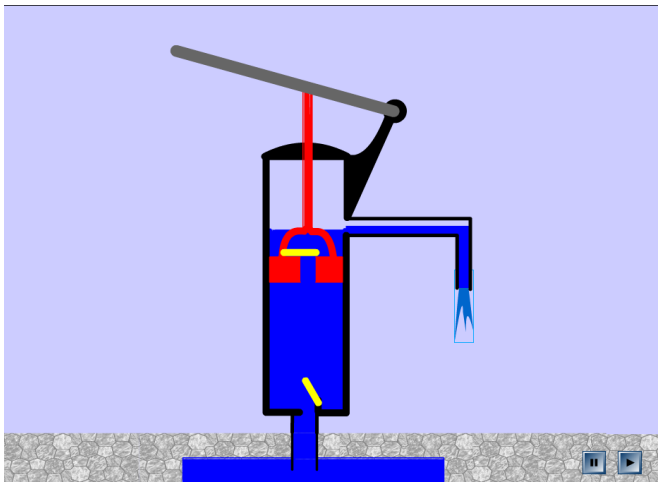
Приблизний вигляд сцени:



12. Принцип дії рідинного насосу

Візуалізація роботи насоса: всмоктування рідини та її подача під тиском, робота клапанів.

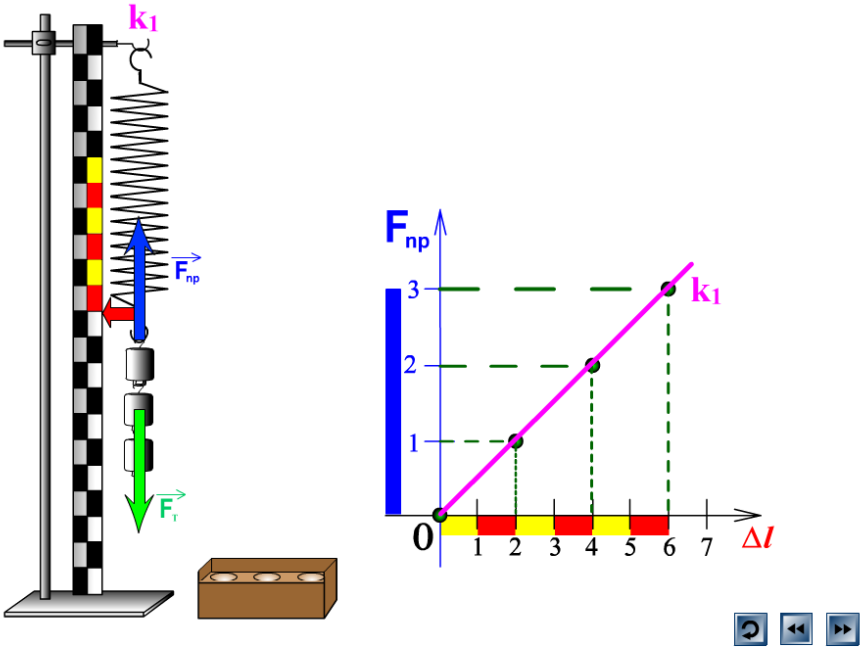
Приблизний вигляд сцени:



13. Закон Гука

Інтерактивна анімація, де можна змінювати видовження пружини шляхом додавання вантажу, для побудови графіку залежності сили пружності від видовження пружини.

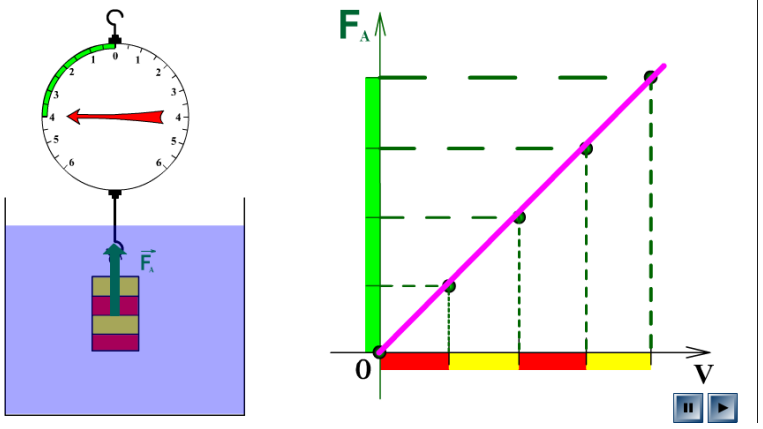
Приблизний вигляд сцени:



14. Виштовхувальна сила

Демонстрація виштовхувальної сили, що діє на тіло у рідині, із залежністю від об’єму зануреної частини тіла.

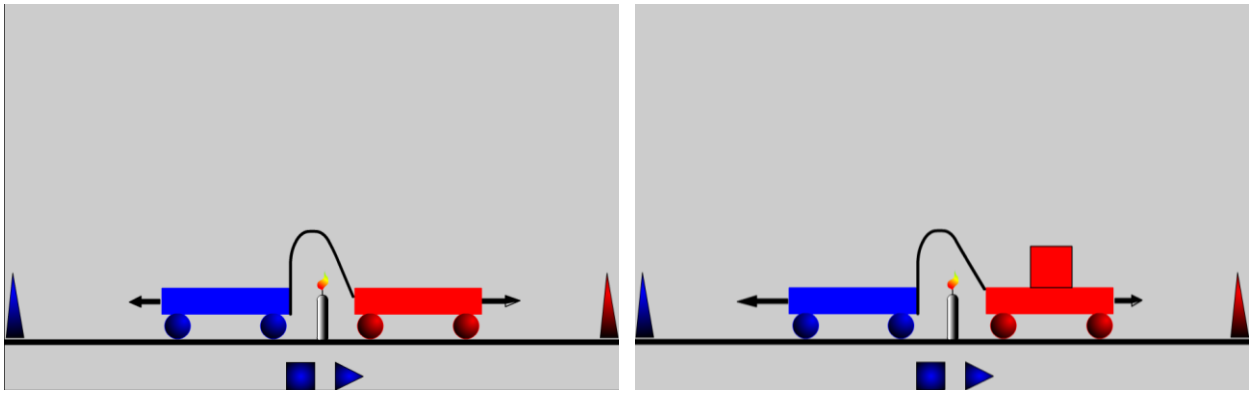
Приблизний вигляд сцени:



15. **Взаємодія візків (вивчення інертності та маси тіла, закону збереження імпульсу)**

Анімація зіткнення двох візків із однаковою (різною) масою із візуалізацією зміни швидкостей тіл під час взаємодії.

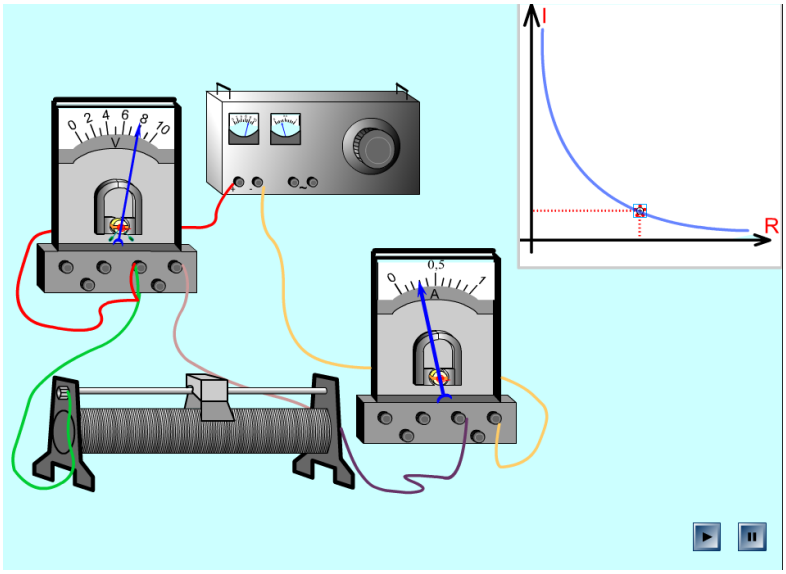
Приблизний вигляд сцени:



16. **Закон Ома (для ділянки кола)**

Показ залежності струму від напруги і опору з можливістю змінювати параметри.

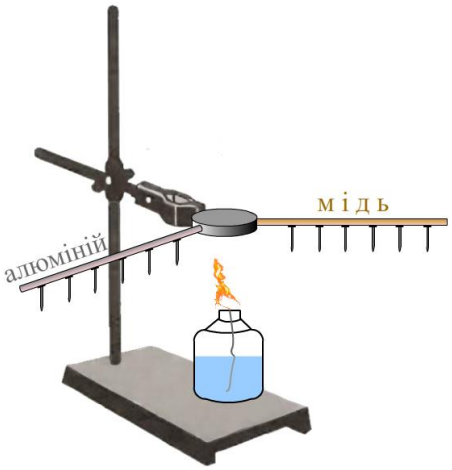
Приблизний вигляд сцени:



17. Теплопровідність у різних матеріалах

Порівняння швидкості передачі тепла через метали з різною теплопровідністю у вигляді анімації.

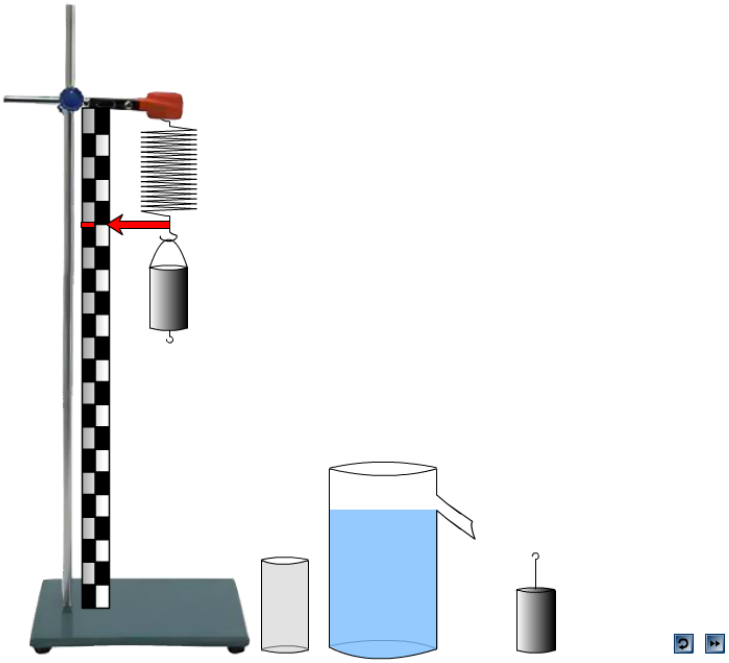
Приблизний вигляд сцени:



18. Закон Архімеда

Демонстрація досліду «Відерце Архімеда». Встановлюється, що вага рідини, витісненої в об'ємі зануреної частини тіла, дорівнює виштовхувальній силі.

Приблизний вигляд сцени:



ActionScript 3.0 (AS3) – це потужна об'єктно-орієнтована мова програмування, яка використовується в Adobe Animate для створення інтерактивних анімацій, навчальних симуляцій, ігор і мультимедійних проєктів. Вона поєднує простоту синтаксису з широкими можливостями для роботи з графікою, подіями, введенням користувача та анімацією.

Цей довідник містить основні поняття, команди, функції, приклади, а також корисні поради для початківців і тих, хто хоче створювати інтерактивні фізичні моделі.

Основи синтаксису ActionScript 3.0

Оголошення змінних

```
var speed:Number = 10.5;      // Дробове число
var count:int = 5;            // Ціле число
var name:String = "Physics";  // Текстовий рядок
var isActive:Boolean = true;  // Логічне значення
```

- var – ключове слово для оголошення змінної.
- Тип змінної вказується через двокрапку.
- Змінні можна оголошувати без початкового значення.

Типи даних

Тип	Опис	Приклад
int	Цілі числа	var x:int = 10;
Number	Числа з плаваючою комою	var pi:Number = 3.14;
String	Текстові рядки	var text:String = "Hi";
Boolean	Логічні значення (true/false)	var flag:Boolean = false;
Array	Масиви	var arr:Array = [];
Object	Загальний об'єкт	var obj:Object = {};

Оператори та вирази

Арифметичні оператори

- + – додавання
- - – віднімання
- * – множення
- / – ділення
- % – остача від ділення

Приклад:

```
var a:int = 10;  // ціле число a=10
var b:int = 3;  // ціле число b=3
```

```
trace(a % b);  // виведення остачі від ділення a/b. Виведе 1
```

Оператори порівняння

- == – рівність
- != – нерівність
- === – сувора рівність (тип і значення)
- !== – сувора нерівність
- > – більше
- < – менше
- >= – більше або дорівнює
- <= – менше або дорівнює

Приклад:

Виконаємо порівняння двох різних типів величин (5 – число, "5" – це рядок):

```
trace(5 == "5");    // true (ActionScript 3.0 перетворить рядок "5" на число 5, і порівняє їх як числа)
trace(5 === "5");  // false (різні типи)
```

Логічні оператори

- && – логічне «і»
- || – логічне «або»
- ! – заперечення

Приклад:

```
var x:Boolean = true;    // оголошує змінну x типу Boolean (логічна) і присвоює їй значення true
var y:Boolean = false;  // оголошує змінну y типу Boolean і присвоює їй значення false.
trace(x && y);  // порівнює і виводить результат false
trace(x || y);  // порівнює і виводить true, якщо хоча б одна величина істинна
trace(!x);      // інвертує логічне значення величини x і виводить false
```

Оператори присвоєння

- = – присвоїти
- += – додати і присвоїти
- -= – відняти і присвоїти
- *= – помножити і присвоїти
- /= – поділити і присвоїти

Приклад:

```
var count:int = 10;  // оголошує змінну count типу int (ціле число) і присвоює їй початкове значення 10
count += 5;  // додає 5 до поточного значення count і виводить count = 15
count *= 2;  // множить поточне значення count на 2 і виводить count = 30
```

Умовні конструкції та цикли

Умовний оператор if

Приклад:

```
if (speed > 10) {  // оператор if (якщо) перевіряє значення змінної speed і
                   виводить відповідне повідомлення залежно від її
                   величини
   trace("Висока швидкість");
} else if (speed > 5) {  // else if (інакше якщо)
   trace("Середня швидкість");
} else {
   trace("Низька швидкість");
}
```

Оператор switch

Приклад:

```
switch(day) {
   case "Monday":
      trace("Понеділок");
      break;
   case "Friday":
      trace("П’ятниця");
      break;
   default:
      trace("Інший день");
}
```

Цей код перевіряє значення змінної day і виводить відповідний текст у консоль залежно від того, яке значення має day.

Значення змінної day порівнюється послідовно з кожним значенням у case.

Якщо знаходиться співпадіння, виконується відповідний блок коду.

Ключове слово break зупиняє виконання switch, щоб не виконувати наступні кейси.

Якщо жоден case не підходить, виконується блок default.

Цикл for

Приклад:

```
for (var i:int = 0; i < 5; i++) {
   trace("Крок: " + i);
}
```

Цей код виконує цикл, який повторюється 5 разів, виводячи у консоль повідомлення з номером кроку.

var i:int = 0; – оголошення змінної і типу int (ціле число), початкове значення 0.

i < 5; – умова продовження циклу: поки і менше 5, цикл триває.

i++ – збільшення змінної і на 1 після кожної ітерації циклу.

Цикл while

Приклад:

```
var i:int = 0;
while (i < 5) {
    trace(i);
    i++;
}
```

Цей код виконує цикл while, який повторюється доти, доки змінна i менша за 5. Під час кожної ітерації виводиться поточне значення i, після чого i збільшується на 1.

var i:int = 0; – оголошення змінної i типу int з початковим значенням 0.

while (i < 5) – умова циклу: поки i менше 5, тіло циклу виконується.

trace(i); – виводить у консоль поточне значення i.

i++; – збільшує i на 1 після кожної ітерації.

Цикл do...while

Приклад:

```
var i:int = 0;
do {
    trace(i);
    i++;
} while (i < 5);
```

Цей код виконує цикл do...while, який гарантує, що тіло циклу виконається принаймні один раз, а потім повторюватиметься доти, доки умова i < 5 є істинною.

Спочатку виконується тіло циклу: виводиться поточне значення i, потім збільшується i на 1.

Після виконання тіла циклу перевіряється умова i < 5.

Якщо умова істинна, цикл повторюється.

Якщо умова хибна – цикл завершується.

Початкове значення i – 0.

Цикл виведе числа від 0 до 4 включно.

Функції

Оголошення функції

Приклад:

```
function greet(name:String):void {
    trace("Привіт, " + name + "!");
}
greet("Світ");
```

Оголошується функція greet, яка приймає один параметр: name:String – рядок, що містить ім'я або будь-який текст.

Функція не повертає значення (void).

Усередині функції виконується вивід у консоль повідомлення: Привіт, <name>! – де <name> – значення параметра name.

greet("Світ"); – викликає функцію greet, передаючи їй рядок "Світ".
У консолі з'явиться: Привіт, Світ!

Приклад:

```
function sum(a:Number, b:Number):Number {
   return a + b;
}
trace(sum(3, 4));
```

Оголошується функція sum, яка приймає два параметри: a:Number – перше число, b:Number – друге число.
Функція повертає значення типу Number – суму цих двох чисел.
В тілі функції виконується операція додавання a + b, і результат повертається оператором return.
trace() виводить у консоль число 7.

Функції з параметрами за замовчуванням

Приклад:

```
function power(base:Number, exponent:int = 2):Number {
   return Math.pow(base, exponent);
}
trace(power(5));
trace(power(5, 3));
```

Оголошується функція power, яка приймає два параметри: base:Number – основа степеня (число, яке підноситься до степеня), exponent:int = 2 – показник степеня (ціле число), за замовчуванням дорівнює 2.
Функція повертає тип Number – результат піднесення base у степінь exponent.
Для обчислення використовується вбудована функція Math.pow(base, exponent).
trace(power(5)); – викликає power з base = 5 і не вказує exponent, тому використовується значення за замовчуванням – 2.
Обчислюється 5^2 = 25. Виводить у консоль: 25.
trace(power(5, 3)); – викликає power з base = 5 і exponent = 3.
Обчислюється 5^3 = 125. Виводить у консоль:125.

Тригонометричні функції

ActionScript 3.0 має вбудовані функції для роботи з тригонометрією в класі Math.

Функція	Опис	Приклад
Math.sin(angle)	Синус кута (в радіанах)	Math.sin(Math.PI / 2); // 1
Math.cos(angle)	Косинус кута	Math.cos(0); // 1
Math.tan(angle)	Тангенс кута	Math.tan(Math.PI / 4); // 1

Функція	Опис	Приклад
Math.asin(value)	Арксинус (повертає радіани)	
Math.acos(value)	Арккосинус	
Math.atan(value)	Арктангенс	

Приклад:

```
var radius:Number = 100;
var angle:Number = 45 * Math.PI / 180;
var x:Number = radius * Math.cos(angle);
var y:Number = radius * Math.sin(angle);
trace("Координати: (" + x + ", " + y + ")");
```

Цей код обчислює координати точки на колі з радіусом 100, розташованої під кутом 45 градусів відносно осі X.

var radius:Number = 100; – визначає радіус кола – відстань від центру до точки.

*var angle:Number = 45 * Math.PI / 180; – перетворює кут з градусів у радіани.*

Оскільки тригонометричні функції (Math.cos, Math.sin) в ActionScript використовують радіани, а не градуси, кут 45° переводиться у радіани за формулою: $\text{радіани} = \text{градуси} \cdot \frac{\pi}{180}$.

*var x:Number = radius * Math.cos(angle); – обчислює X-координату точки на колі.*

*var y:Number = radius * Math.sin(angle); – обчислює Y-координату точки на колі.*

trace("Координати: (" + x + ", " + y + ")"); – виводить у консоль координати точки у форматі: Координати: (x, y).

Для кута 45 градусів і радіуса 100 координати приблизно будуть:
Координати: (70.71067811865476, 70.71067811865474).

Малювання ліній та фігур

Використання класу Graphics

Для малювання графіки використовують властивість graphics об’єктів типу Shape або Sprite.

Основні методи

Метод	Опис	Приклад
lineStyle(thickness, color, alpha)	Встановлення товщини, кольору і прозорості лінії	graphics.lineStyle(2, 0xFF0000, 1);
moveTo(x, y)	Початок лінії	graphics.moveTo(10, 10);
lineTo(x, y)	Малювання лінії до точки	graphics.lineTo(100, 100);
beginFill(color, alpha)	Початок заливки	graphics.beginFill(0x00FF00, 0.5);
drawRect(x, y, width, height)	Малювання прямокутника	graphics.drawRect(10, 10, 100, 50);
drawCircle(x, y, radius)	Малювання кола	graphics.drawCircle(50, 50, 30);
endFill()	Завершення заливки	graphics.endFill();

Приклад:

```
var shape:Shape = new Shape();
shape.graphics.lineStyle(3, 0x0000FF);
shape.graphics.beginFill(0xFFFF00, 0.7);
shape.graphics.drawRect(50, 50, 150, 100);
shape.graphics.endFill();
addChild(shape);
```

Цей код створює графічний об'єкт (прямокутник) із синім контуром і напівпрозорою жовтою заливкою, а потім додає його на сцену.
var shape:Shape = new Shape(); – створює новий об'єкт типу Shape, який використовується для малювання векторної графіки.
shape.graphics.lineStyle(3, 0x0000FF); – встановлює стиль лінії для майбутніх фігур: 3 – товщина лінії (3 пікселі), 0x0000FF – колір лінії (синій, у шістнадцятковому форматі RGB).
shape.graphics.beginFill(0xFFFF00, 0.7); – вказує, що наступна фігура буде залита кольором: 0xFFFF00 – колір заливки (жовтий), 0.7 – прозорість заливки (від 0 до 1, де 1 – повністю непрозорий, 0.7 – 70% непрозорості).
shape.graphics.drawRect(50, 50, 150, 100); – малює прямокутник: 50, 50 – координати верхнього лівого кута (відносно контейнера), 150 – ширина прямокутника, 100 – висота прямокутника.
shape.graphics.endFill(); – завершує заливку фігури.
addChild(shape); – додає створений об'єкт shape на сцену, щоб він став видимим. На сцені з'явиться прямокутник: з координатами верхнього лівого кута (50, 50), розміром 150×100 пікселів, із синім контуром

товщиною 3 пікселі, із напівпрозорою жовтою заливкою (70% непрозорості).

Зміна кольору і товщини ліній

- Колір задається у шістнадцятковому форматі, наприклад, 0xFF0000 – червоний.
- Товщина – число пікселів.
- Прозорість (alpha) – від 0 (прозорий) до 1 (непрозорий).

Приклад:

```
graphics.lineStyle(5, 0x00FF00, 0.8); // Зелена лінія товщиною 5 пікселів, прозорість 0.8
```

Побудова синусоїди

```
import flash.display.Shape;
var sineWave:Shape = new Shape();
addChild(sineWave);
var amplitude:Number = 50;
var frequency:Number = 0.05;
var offsetY:Number = 200;
sineWave.graphics.lineStyle(2, 0x0000FF);
sineWave.graphics.moveTo(0, offsetY);
for (var x:int = 0; x <= stage.stageWidth; x++) {
var y:Number = amplitude * Math.sin(x * frequency) + offsetY;
sineWave.graphics.lineTo(x, y); }
```

Цей код на ActionScript 3.0 малює графік синусоїди на сцені за допомогою об'єкта Shape і його графічного контексту graphics.

import flash.display.Shape; – імпортується клас Shape для створення векторної графіки.

var sineWave:Shape = new Shape(); addChild(sineWave); – створюється новий об'єкт sineWave типу Shape і додається на сцену, щоб він був видимим.

var amplitude:Number = 50; – амплітуда синусоїди, тобто максимальне відхилення по вертикалі (висота коливань).

var frequency:Number = 0.05; – частота синусоїди, визначає, як часто коливання повторюються по горизонталі (чим більше значення, тим частіше).

var offsetY:Number = 200; – вертикальний зсув графіка, щоб синусоїда була розташована не на самому низу, а приблизно посередині по осі Y.

sineWave.graphics.lineStyle(2, 0x0000FF); – встановлюється товщина лінії 2 пікселі і синій колір (0x0000FF).

sineWave.graphics.moveTo(0, offsetY); – починається малювання лінії з точки (0, offsetY) — зсув по вертикалі.

for (var x:int = 0; x <= stage.stageWidth; x++) { – цикл проходить по всій ширині сцени (stage.stageWidth).

*var y:Number = amplitude * Math.sin(x * frequency) + offsetY; – для кожної*

координати x обчислюється y за формулою синуса.
sineWave.graphics.lineTo(x, y); – за допомогою lineTo малюється лінія, що проходить через всі обчислені точки, утворюючи графік синусоїди.
Отже, код малює плавну синусоїду, розтягнуту по ширині сцени. Амплітуда та частота керують формою хвилі. Зсув offsetY піднімає графік, щоб він не був на самому низу.

Відстеження натискання клавіш клавіатури

Додавання слухачів подій клавіатури

Приклад:

```
stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);
stage.addEventListener(KeyboardEvent.KEY_UP, onKeyUp);

function onKeyDown(event:KeyboardEvent):void {
   trace("Натиснута клавіша: " + event.keyCode);
}
function onKeyUp(event:KeyboardEvent):void {
   trace("Відпущена клавіша: " + event.keyCode);
}
```

Цей код додає обробники подій для відслідковування натискань і відпускань клавіш на клавіатурі під час роботи Flash-програми.
stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown); – реєструє слухача події KEY_DOWN (натискання клавіші) на об'єкті stage (головній сцені).Коли користувач натискає клавішу, викликається функція onKeyDown.
stage.addEventListener(KeyboardEvent.KEY_UP, onKeyUp); – реєструє слухача події KEY_UP (відпускання клавіші). Коли користувач відпускає клавішу, викликається функція onKeyUp.
Функції обробки подій:
onKeyDown(event:KeyboardEvent):void – параметр event містить інформацію про подію клавіатури, виводить у консоль повідомлення з кодом натиснутої клавіші: Натиснута клавіша: <keyCode>.
event.keyCode – числовий код клавіші (наприклад, 65 для 'A').
onKeyUp(event:KeyboardEvent):void – аналогічно, виводить повідомлення про відпущену клавішу:Відпущена клавіша: <keyCode>.
Якщо натиснути клавішу "A" (код 65), у консолі з'явиться: Натиснута клавіша: 65. Після відпускання клавіші: Відпущена клавіша: 65.

Коди основних клавіш

Клавіша	Код (keyCode)
Стрілка вгору	38
Стрілка вниз	40

Клавіша	Код (keyCode)
Стрілка вліво	37
Стрілка вправо	39
Пробіл	32
Enter	13

Приклад:

```
var speed:Number = 5;

stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);

function onKeyDown(e:KeyboardEvent):void {
   switch(e.keyCode) {
      case 37: myBall.x -= speed; break; // Вліво
      case 39: myBall.x += speed; break; // Вправо
      case 38: myBall.y -= speed; break; // Вгору
      case 40: myBall.y += speed; break; // Вниз
   }
}
```

Цей код дозволяє керувати позицією об'єкта myBall на сцені за допомогою клавіш-стрілок на клавіатурі.
var speed:Number = 5; – задає швидкість переміщення об'єкта – на скільки пікселів змінюється координата при кожному натисканні клавіші.
stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown); – додає обробник події натискання клавіші на сцені.
Функція onKeyDown(e:KeyboardEvent):void – виконується при кожному натисканні клавіші. В залежності від коду натиснутої клавіші (e.keyCode) змінює позицію об'єкта myBall.

Переміщення об’єктів мишею

Основні події миші

Подія	Опис
MouseEvent.MOUSE_DOWN	Натискання кнопки миші
MouseEvent.MOUSE_UP	Відпускання кнопки
MouseEvent.MOUSE_MOVE	Рух миші
MouseEvent.CLICK	Клік миші

Перетягування об'єкта

Приклад:

```
myBall.addEventListener(MouseEvent.MOUSE_DOWN, startDragBall);
stage.addEventListener(MouseEvent.MOUSE_UP, stopDragBall);

function startDragBall(e:MouseEvent):void {
   myBall.startDrag();
}

function stopDragBall(e:MouseEvent):void {
   myBall.stopDrag();
}
```

Цей код дозволяє користувачу перетягувати об'єкт myBall мишею на сцені.

myBall.addEventListener(MouseEvent.MOUSE_DOWN, startDragBall); – додає обробник події натискання кнопки миші на об'єкті myBall. Коли користувач натискає мишу на myBall, викликається функція startDragBall.

stage.addEventListener(MouseEvent.MOUSE_UP, stopDragBall); – додає обробник події відпускання кнопки миші на всій сцені (stage). Коли користувач відпускає кнопку миші будь-де на сцені, викликається функція stopDragBall.

Функція startDragBall(e:MouseEvent):void – викликає метод startDrag() у об'єкта myBall, що починає перетягування об'єкта за курсором миші.

Функція stopDragBall(e:MouseEvent):void – викликає метод stopDrag() у об'єкта myBall, що припиняє перетягування.

Коли користувач натискає і утримує кнопку миші на об'єкті myBall, об'єкт "прилипає" до курсору і рухається разом з ним. Коли користувач відпускає кнопку миші, перетягування припиняється, і об'єкт залишається у новому положенні.

Слідування за курсором миші

Приклад:

```
stage.addEventListener(MouseEvent.MOUSE_MOVE, onMouseMove);

function onMouseMove(e:MouseEvent):void {
   myBall.x = e.stageX;
   myBall.y = e.stageY;
}
```

Цей код змушує об'єкт myBall постійно слідувати за курсором миші на сцені.

stage.addEventListener(MouseEvent.MOUSE_MOVE, onMouseMove); – додає обробник події руху миші на сцені. Кожен раз, коли користувач рухає мишу, викликається функція onMouseMove.

Функція onMouseMove(e:MouseEvent):void – встановлює позицію

об'єкта myBall у координати курсора миші: e.stageX – горизонтальна координата курсора на сцені, e.stageY – вертикальна координата курсора на сцені.
Об'єкт myBall миттєво переміщується у позицію курсора миші, створюючи ефект "прилипання" до курсора.

Приклади корисних команд і конструкцій

Вивід у консоль

Приклад:

```
trace("Повідомлення для відладки");
```

Таймер

Приклад:

```
import flash.utils.Timer;
import flash.events.TimerEvent;

var timer:Timer = new Timer(1000, 5); // 1 секунда, 5 повторів
timer.addEventListener(TimerEvent.TIMER, onTick);
timer.start();

function onTick(e:TimerEvent):void {
    trace("Тік таймера");
}
```

Цей код створює таймер, який спрацьовує кожну секунду (1000 мілісекунд) і виконує подію 5 разів, виводячи повідомлення у консоль.
Імпорти:
flash.utils.Timer – клас таймера.
flash.events.TimerEvent – події, що генеруються таймером.
var timer:Timer = new Timer(1000, 5); – створює новий об'єкт таймера: 1000 – інтервал у мілісекундах (1 секунда), 5 – кількість повторів (таймер спрацює 5 разів).
timer.addEventListener(TimerEvent.TIMER, onTick); – додає слухача події TIMER, яка викликається кожного разу, коли спливає інтервал.
timer.start(); – запускає таймер.
Функція onTick(e:TimerEvent):void – виконується кожного разу, коли таймер "тікає" (через кожну секунду).Виводить у консоль повідомлення: Тік таймера.

Перевірка типу

Приклад:

```
if (myVar is Number) {
    trace("Це число");
}
```

Обробка помилок

Приклад:

```
try {
   var result:Number = 10 / 0;
} catch (e:Error) {
   trace("Помилка: " + e.message);
}
```

Масиви і цикл for each

```
var arr:Array = [1, 2, 3, 4, 5];
for each (var num:int in arr) {
   trace(num);
}
```

Оголошується масив arr, що містить числа від 1 до 5.
Виконує цикл for each, який перебирає кожен елемент масиву.
Для кожного елемента змінна num приймає значення елемента і виводить його у консоль за допомогою trace().

Робота з об’єктами на сцені

Зміна властивостей

Приклад:

```
myClip.x = 100;
myClip.y = 200;
myClip.rotation = 45;
myClip.alpha = 0.5;
```

Цей код встановлює позицію, обертання та прозорість об'єкта myClip на сцені.
myClip.x = 100; – встановлює горизонтальну координату об'єкта myClip в 100 пікселів від лівого краю батьківського контейнера.
myClip.y = 200; – встановлює вертикальну координату об'єкта myClip в 200 пікселів від верхнього краю батьківського контейнера.
myClip.rotation = 45; – повертає об'єкт myClip на 45 градусів за годинниковою стрілкою навколо його точки трансформації (за замовчуванням – центр об'єкта).
myClip.alpha = 0.5; – встановлює прозорість об'єкта myClip. Значення 0.5 означає 50% прозорість (півпрозорий).

Додавання і видалення об’єктів

Приклад:

```
addChild(myClip);    // Додати на сцену
removeChild(myClip); // Видалити зі сцени
```

Приклад класу:

```
package {
   import flash.display.MovieClip;
   import flash.events.Event;
   public class Ball extends MovieClip {
      public var speed:Number = 5;

      public function Ball() {
         addEventListener(Event.ENTER_FRAME, move);
      }

      private function move(e:Event):void {
         this.x += speed;
      }
   }
}
```

Цей код описує клас Ball, який наслідує від MovieClip і реалізує просту анімацію руху об'єкта вправо.

Пакет і імпорти:

package { ... } – оголошення пакету (у цьому випадку без імені, тобто кореневий пакет).

Імпортуються класи MovieClip (для графічного об'єкта) та Event (для обробки подій).

Клас Ball: наслідується від MovieClip, тому може бути доданий на сцену як графічний об'єкт. Має публічну змінну speed типу Number, що визначає швидкість руху (за замовчуванням 5).

Конструктор Ball(): викликається при створенні об'єкта класу, додає слухача події Event.ENTER_FRAME, яка викликається кожного кадру анімації (зазвичай 24 або 30 разів на секунду).

Обробником події є метод move. Метод move(e:Event):void: виконується кожного кадру, збільшує координату x об'єкта на значення speed, тим самим рухаючи об'єкт вправо.

Використання класу

Приклад:

```
var ball:Ball = new Ball();
addChild(ball);
ball.x = 100;
ball.y = 200;
```

Створюється новий об'єкт ball класу Ball. Додається цей об'єкт на сцену, щоб він став видимим. Встановлюється позиція об'єкта ball на координати (100, 200).

var ball:Ball = new Ball(); – створює екземпляр класу Ball. При цьому виконується конструктор класу, який, наприклад, може запускати

анімацію або налаштовувати об'єкт.
addChild(ball); – додає об'єкт ball до дисплейного списку сцени, роблячи його видимим і активним.
ball.x = 100; і ball.y = 200; – встановлюють координати об'єкта ball відносно верхнього лівого кута батьківського контейнера (зазвичай сцени).

Цей довідник охоплює основи ***ActionScript 3.0*** – від синтаксису і типів даних до роботи з графікою, подіями, клавіатурою і мишею. Знання цих інструментів допоможе вам створювати інтерактивні, динамічні моделі фізичних явищ, які зроблять навчання живим і зрозумілим. Практикуйтеся, експериментуйте, і ви зможете створювати власні навчальні проєкти, які надихатимуть ваших учнів пізнавати фізику з цікавістю і захопленням!

Корисні посилання

Відеоуроки українською

- ***Adobe Animate. Класична анімація руху. Розгін-гальмування***

  Короткий практичний урок для гуртка «Комп'ютерна графіка». Демонструє основи класичної анімації руху, принципи розгону та гальмування, підходить для початківців.

- ***Як зробити свій перший мультик (Adobe Animate)***

  Покрокове відео для новачків: від ідеї до готової анімації. Охоплює вибір ідеї, сторіборд, налаштування програми, створення аніматиків, роботу зі звуком і рендеринг. Все пояснюється простою українською мовою.

- ***Скелетна анімація Adobe Animate. Інструмент «Кістка»***

  Практичний урок для гуртківців: як працювати з інструментом «Кістка» для створення скелетної анімації. Пояснюється на прикладах, підходить для тих, хто вже знайомий з базовими інструментами.

- ***Уроки по Adobe Animate (плейлист)***

  Підбірка відеоуроків з різних аспектів роботи в Adobe Animate: створення гіфок, анімація персонажів, покадрова анімація.

Електронні довідники, посібники, методичні матеріали

- ***Офіційна сторінка Adobe Animate українською***

  Опис можливостей програми, посилання на навчальні матеріали, пробна версія, підтримка користувачів.

- ***Навчальні матеріали та підтримка від Adobe***

  Україномовний розділ із посібниками, відповідями на часті питання, форумом спільноти та інструкціями для початківців і досвідчених користувачів.

- ***Методична доповідь: «Використання флеш анімацій та комп'ютерних програм при вивченні фізики»***

  Описує педагогічні підходи до використання анімацій у фізиці, містить приклади уроків, поради для вчителів, як інтегрувати анімацію у навчальний процес.

- ***Курс «Основи мультиплікації в Adobe Animate»***

  Україномовний онлайн-курс для дітей і дорослих: від основ інтерфейсу до створення власних мультфільмів, інтерактивних ігор та анімованих сторінок.

- ***Методичні рекомендації з комп'ютерної анімації***

  Документ містить вступ до анімації, історію, принципи використання растрової та векторної графіки, а також методичні поради щодо створення комп'ютерної анімації, зокрема з використанням Adobe Flash/Animate.

- ***Використання флеш-анімацій та комп'ютерних програм при вивченні фізики***

  Описуються педагогічні аспекти використання флеш-анімацій у вивченні фізики, з прикладами застосування анімації для візуалізації фізичних процесів і підвищення мотивації здобувачів.

- ***Анімаційна графіка та motion дизайн***

  Розглядається програмне забезпечення для створення анімації, зокрема Adobe Animate, та їх застосування у навчальному процесі, з прикладами і методичними рекомендаціями.

Довідники та навчальні матеріали з ActionScript 3.0

- ***Офіційна документація та навчальні матеріали Adobe (англійською)***

  Вичерпний довідник з ActionScript 3.0, приклади коду, пояснення основних класів, подій, обробки графіки та анімації.

- ***Learning ActionScript 3.0 (PDF, англійською)***

  Огляд основних можливостей мови, синтаксису, особливостей роботи з подіями, графікою, текстом, API.

Платформи, додатки та бібліотеки анімацій

- ***Візуальна фізика: Анімації (додаток для Android)***

  Велика бібліотека фізичних анімацій і відео українською. Дозволяє вивчати фізику через інтерактивні моделі, підходить для учнів, здобувачів і вчителів.

Україномовні спільноти, форуми, обговорення

- ***DOU: Програмістам ActionScript 3.0***

  Український форум для обговорення питань програмування на ActionScript 3.0, обмін досвідом, поради щодо вирішення типових проблем, пошук однодумців.

- ***Фейсбук-групи та спільноти для вчителів фізики***

  Варто шукати за тегами: #AdobeAnimate #АнімаціяУкраїнською

#ВикладанняФізики. Тут часто діляться власними напрацюваннями, відеоуроками, методичними матеріалами.

Офіційні ресурси Adobe

- ***Adobe Animate: навчальні матеріали, пробна версія, підтримка***

  Офіційний сайт з україномовним інтерфейсом, де можна завантажити пробну версію, знайти інструкції, відеоуроки, відповіді на часті питання.

- ***Learning ActionScript 3.0 (Adobe, англ.)***

  Офіційний посібник для самостійного вивчення ActionScript 3.0 з прикладами, поясненнями, структурованими розділами для різних рівнів підготовки.

- ***Complete Adobe Animate Megacourse: Beginner to Expert (Udemy, англ.)***

  Відеокурс для початківців і тих, хто хоче поглибити знання. Охоплює всі основи роботи з Adobe Animate, принципи анімації, створення персонажів, інтерактивних сцен.

Список літератури

1. Adobe Animate. Класична анімація руху. Розгін-гальмування [Відеоурок]. URL: https://www.youtube.com/watch?v=...
2. Bell P., Linn M.C. Scientific arguments as learning artifacts: Designing for learning from the web with KIE // International Journal of Science Education. 2000. Vol. 22, № 8. С. 797–817.
3. Bell T., Urhahne D., Schanze S., Ploetzner R. Collaborative inquiry learning: Models, tools, and challenges // International Journal of Science Education. 2010. Vol. 32, № 3. С. 349–377.
4. Blikstein P. Digital fabrication and ‘making’ in education: The democratization of invention // FabLabs: Of Machines, Makers and Inventors. 2014. С. 1–21.
5. Blikstein P., Kabayadondo Z., Martin A. та ін. The Maker Movement in Education: A Literature Review. Stanford: Stanford University, 2017.
6. Blikstein P. Maker movement in education: History and prospects // Holman V. (Ed.), The Cambridge Handbook of Computing Education Research. Cambridge: Cambridge University Press, 2019.
7. Биков В.Ю., Шишкіна М.П. Хмарні сервіси в освіті: теорія і практика використання. Київ: Інститут інформаційних технологій і засобів навчання НАПН України, 2014.
8. Бондаренко О.Г. Інформаційно-комунікаційні технології у навчанні фізики: навчальний посібник. Київ: Ліра-К, 2018.
9. Bransford J.D., Brown A.L., Cocking R.R. How People Learn: Brain, Mind, Experience, and School. Washington: National Academy Press, 2000.
10. Chao C.Y., Chou H.W., Chen Y.T. Using interactive simulations in physics education: The impact of prior knowledge and cognitive load // Computers & Education. 2017. Vol. 115. С. 14–27.
11. Clark D.B., Nelson B.C., Sengupta P., D’Angelo C.M. Rethinking science learning through digital games and simulations. Washington: National Academies Press, 2009.
12. Clark R.C., Mayer R.E. E-Learning and the Science of Instruction. Hoboken: Wiley, 2016.
13. de Jong T., Linn M.C., Zacharia Z.C. Physical and virtual laboratories in science and engineering education // Science. 2013. Vol. 340, № 6130. С. 305–308.
14. Dede C. Digital teaching platforms: Customizing classroom learning for each student. New York: Teachers College Press, 2011.
15. Dori Y.J., Belcher J. How does technology-enabled active learning affect undergraduate students’ understanding of electromagnetism concepts? // The Journal of the Learning Sciences. 2005. Vol. 14, № 2. С. 243–279.
16. Довгий С.О., Козяр М.М. Інтерактивні технології у навчанні фізики. Тернопіль: ТНПУ, 2021.
17. Ерстад О. Digital kompetanse i skolen. Oslo: Universitetsforlaget, 2010.

18. European Commission. Digital Competence Framework for Citizens (DigComp 2.1). Luxembourg: Publications Office of the European Union, 2017.
19. European Commission. Digital Education Action Plan 2021–2027. Luxembourg: Publications Office of the European Union, 2021.
20. European Commission. Ethical guidelines for trustworthy AI. Brussels: European Commission, 2019.
21. European Commission. Innovating Education and Educating for Innovation. Luxembourg: Publications Office of the European Union, 2016.
22. European Commission. Key Competences for Lifelong Learning. Luxembourg: Publications Office of the European Union, 2019.
23. European Commission. Opening up Education: Innovative teaching and learning for all through new Technologies and Open Educational Resources. Brussels: European Commission, 2013.
24. European Commission. Selfie for Teachers: Digital Competence Self-Reflection Tool for Primary and Secondary School Teachers. Luxembourg: Publications Office of the European Union, 2021.
25. European Commission. Shaping Europe's digital future. Brussels: European Commission, 2020.
26. European Commission. Supporting teacher competence development for better learning outcomes. Luxembourg: Publications Office of the European Union, 2013.
27. European Schoolnet. KeyCoNet: Key Competence Network on School Education. Brussels: European Schoolnet, 2014.
28. European Schoolnet. The Future Classroom Lab Toolkit. Brussels: European Schoolnet, 2018.
29. Finkelstein N.D., Adams W.K., Keller C.J. та ін. When learning about the real world is better done virtually: A study of substituting computer simulations for laboratory equipment // Physical Review Special Topics – Physics Education Research. 2005. Vol. 1, № 1. 010103.
30. Fullan M. The New Meaning of Educational Change. New York: Teachers College Press, 2016.
31. Fullan M., Langworthy M. A Rich Seam: How New Pedagogies Find Deep Learning. London: Pearson, 2014.
32. Gee J.P. What Video Games Have to Teach Us About Learning and Literacy. New York: Palgrave Macmillan, 2007.
33. Герасименко І.В., Коноваленко О.Р., Точинська Я.О. Використання мультимедійних ресурсів у навчальному процесі (на прикладі Adobe Flash Professional) // Інформаційні технології і засоби навчання. 2017. № 4(60). С. 76–85.
34. Гринюк В.І. Використання інформаційних технологій у навчанні фізики. Чернівці: ЧНУ, 2017.
35. Гуржій А.М., Семеріков С.О. Інформаційно-комунікаційні технології в освіті. Київ: Педагогічна думка, 2011.

36. Гуревич Р.С. Візуалізація навчального матеріалу з фізики. Київ: Ліра-К, 2019.
37. Hattie J. Visible Learning: A Synthesis of Over 800 Meta-Analyses Relating to Achievement. London: Routledge, 2009.
38. Hattie J. Visible Learning for Teachers: Maximizing Impact on Learning. London: Routledge, 2012.
39. Hegedus S.J., Dalton S., Tapper J. The impact of technology-enhanced curriculum on learning advanced algebra in US high school classrooms // Educational Technology Research and Development. 2015. Vol. 63, № 2. С. 203–228.
40. Hennessy S., Deaney R., Ruthven K. Emerging teacher strategies for supporting subject teaching and learning with ICT // Curriculum Journal. 2006. Vol. 17, № 2. С. 139–166.
41. Hsin W.J., Cigas J. Short videos improve student learning in online education // Journal of Computing Sciences in Colleges. 2013. Vol. 28, № 5. С. 253–259.
42. Захарченко О.О. Використання задач-анімацій в освітньому процесі з фізики на етапі загальної середньої освіти // Фізика та астрономія в школі. 2019. № 4. С. 22–28.
43. Інчук О.О. Інтерактивні комп'ютерні моделі на уроках фізики: досвід, проблеми, перспективи // Фізика та астрономія в школі. 2020. № 1. С. 14–17.
44. Johnson L., Adams Becker S., Cummins M., Estrada V. NMC Horizon Report: 2013 K-12 Edition. Austin, Texas: The New Media Consortium, 2013.
45. Johnson L., Adams Becker S., Estrada V., Freeman A. NMC Horizon Report: 2014 Higher Education Edition. Austin, Texas: The New Media Consortium, 2014.
46. Jonassen D.H. Computers as Mindtools for Schools. Columbus: Merrill Prentice Hall, 2000.
47. Каленик М.В. Використання комп'ютера на уроках фізики в основній школі // Проблеми методики викладання фізики на сучасному етапі: збірник статей. – Кіровоград: РВЦ КДПУ ім. В. Винниченка, 2000. – С. 46 – 49.
48. Каленик М.В. Комп'ютерні демонстрації під час вивчення технічних пристроїв // Фізика та астрономія в школі. - 2006. - № 4. - С. 50 – 54.
49. Каленик М.В. Внутрішні зворотні зв'язки під час роботи учнів з персональними комп'ютерами як навчальними пристроями // Збірник наукових праць Уманського державного педагогічного університету імені П. Тичини /Гол ред.: Мартинюк М.Т. – Умань: СПД Жовтий, 2008. – Ч.2.– С. 180 – 187.
50. Каленик М.В., Пасько О.О. Методика віртуального демонстраційного фізичного експерименту // Фізика та астрономія в школі. – 2009- № 1 – С.29 -32.

51. Каленик М.В. Формування інформаційної компетентності майбутнього учителя фізики // Актуальні питання природничо-математичної освіти:/ зб. наук. пр.: випуск 2 /Сум. держ. пед. ун-т ім. А.С. Макаренка. – Суми: ВВП "Мрія", 2013. С.143-148.
52. Каленик М.В. Повторення раніше вивченого, перевірка й облік знань і умінь учнів з використанням хмарних технологій / М. В. Каленик // Фізико-математична освіта. - 2017. - Вип. 4. - С. 180-185.
53. Каленик М.В. Організація та проведення дистанційного навчання фізики: досвід, проблеми, перспективи // Наукові записки. Серія: Проблеми методики фізико-математичної та технологічної освіти. 2021. Вип. 25. С. 45–51.
54. Каплун О.Л. Візуалізація навчального матеріалу з фізики засобами ІКТ. Тернопіль: ТНПУ, 2020.
55. Козяр М.М. Методика використання мультимедійних засобів у навчанні фізики. Тернопіль: ТНПУ, 2018.
56. Козяр М.М. Методика використання мультимедійних засобів у навчанні фізики: навчальний посібник. Тернопіль: ТНПУ, 2018.
57. Козявкін С.О., Слободяник О.В. Комп'ютерні симуляції при вивченні атомної фізики у ЗЗСО // Фізика та астрономія в школі. 2019. № 5. С. 22–26.
58. Крамаров С.С. Основи комп'ютерної анімації. Київ: Видавничий дім "Слово", 2017.
59. Кропивний С., Слободяник О.В. Комп'ютерні симуляції при вивченні атомної фізики у ЗЗСО // Фізика та астрономія в школі. 2019. № 5. С. 22–26.
60. Кузьменок Т.С. Використання флеш-анімацій та комп'ютерних програм при вивченні фізики // Фізика та астрономія в школі. 2018. № 2. С. 12–17.
61. Кузьменок Т.С. Методична доповідь: «Використання флеш-анімацій та комп'ютерних програм при вивченні фізики». URL: https://osvita.ua/school/method/...
62. Кузьменок Т.С. Використання флеш-анімацій та комп'ютерних програм при вивченні фізики: методичні рекомендації. URL: https://osvita.ua/school/method/...
63. Кузьменок Т.С. Методичні рекомендації з комп'ютерної анімації. URL: https://osvita.ua/school/method/...
64. Laurillard D. Rethinking University Teaching: A Conversational Framework for the Effective Use of Learning Technologies. London: Routledge, 2002.
65. Laurillard D. Teaching as a Design Science. New York: Routledge, 2012.
66. Linn M.C., Davis E.A., Bell P. Internet environments for science education. Mahwah: Lawrence Erlbaum, 2004.
67. Linn M.C., Eylon B.S. Science Learning and Instruction: Taking Advantage of Technology to Promote Knowledge Integration. New York: Routledge, 2011.

68. Литвинова С.Г. Мультимедійні засоби навчання у сучасній школі. Київ: Педагогічна думка, 2015.
69. Мельник А.Ю. Комп'ютерна анімація у навчанні фізики. Львів: ЛНУ, 2019.
70. Мельничук О.Г. Інформаційні технології в освіті: теорія і практика. Київ: Освіта України, 2018.
71. Методика і теорія застосування ЕОМ у процесі вивчення фізики у педагогічних університетах: навчальний посібник / За ред. С.О. Семерікова. Кривий Ріг: КДПУ, 2012.
72. Moreno R., Mayer R.E. Interactive multimodal learning environments // Educational Psychology Review. 2007. Vol. 19, № 3. С. 309–326.
73. МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ. Використання ІКТ для підвищення ефективності процесу навчання фізики: методичні рекомендації. Київ: МОН України, 2020.
74. Новікова С.О. Комп'ютерне моделювання як чинник формування в учнів фундаментальних знань з фізики // Фізика та астрономія в школі. 2019. № 3. С. 19–23.
75. OECD. Innovating Education and Educating for Innovation: The Power of Digital Technologies and Skills. Paris: OECD Publishing, 2016.
76. OECD. Measuring Innovation in Education 2019. Paris: OECD Publishing, 2019.
77. OECD. PISA 2022 Results (Volume I): Student Performance in Mathematics, Reading and Science. Paris: OECD Publishing, 2023.
78. OECD. Skills for a Digital World. Paris: OECD Publishing, 2016.
79. OECD. Skills Outlook 2019: Thriving in a Digital World. Paris: OECD Publishing, 2019.
80. OECD. The Future of Education and Skills: Education 2030. Paris: OECD Publishing, 2018.
81. OECD. The OECD Handbook for Innovative Learning Environments. Paris: OECD Publishing, 2017.
82. OECD. Trends Shaping Education 2019. Paris: OECD Publishing, 2019.
83. Papert S. Mindstorms: Children, Computers, and Powerful Ideas. New York: Basic Books, 1993.
84. Passey D. Inclusive technology enhanced learning. New York: Routledge, 2013.
85. Papert S. Mindstorms: Children, Computers, and Powerful Ideas. New York: Basic Books, 1993.
86. Prensky M. Digital Game-Based Learning. New York: McGraw-Hill, 2001.
87. Redecker C. European Framework for the Digital Competence of Educators: DigCompEdu. Luxembourg: Publications Office of the European Union, 2017.
88. Redecker C., Punie Y. European Framework for the Digital Competence of Educators: DigCompEdu. Luxembourg: Publications Office of the European Union, 2017.

89. Rutten N., van Joolingen W.R., van der Veen J.T. The learning effects of computer simulations in science education // Computers & Education. 2012. Vol. 58, № 1. С. 136–153.
90. Савчук-Баловсяк Г.Д. Використання мультимедійних технологій на уроках фізики: методична розробка // Фізика та астрономія в школі. 2020. № 6. С. 10–15.
91. Сальник І.В. Мобільні пристрої та сучасне освітнє програмне забезпечення у навчанні фізики в закладах загальної середньої освіти // Фізика та астрономія в школі. 2021. № 2. С. 18–23.
92. Сайт для вчителів фізики. Застосування анімації у навчанні фізики. URL: https://physicsteacher.com.ua/animation
93. Сайт. Застосування комп'ютерних симуляцій на уроках фізики під час дистанційного навчання. URL: https://osvita.ua/school/method/...
94. Сайт. Практичні матеріали, відеоуроки, методичні рекомендації з використання Adobe Animate, Flash, мультимедійних технологій у фізичній освіті. URL: https://osvita.ua/school/method/...
95. Семеріков С.О., Теплицький І.О. Інформаційні технології навчання фізики у вищій школі. Кривий Ріг: КДПУ, 2013.
96. Семеріков С.О., Теплицький І.О. Хмарні технології в освіті. Кривий Ріг: КДПУ, 2015.
97. Smetana L.K., Bell R.L. Computer simulations to support science instruction and learning: A critical review of the literature // International Journal of Science Education. 2012. Vol. 34, № 9. С. 1337–1370.
98. Tatar E., Akkaya A. The effect of computer-assisted physics instruction on achievement and attitudes // Computers & Education. 2017. Vol. 113. С. 16–29.
99. UNESCO. Artificial Intelligence in Education: Challenges and Opportunities for Sustainable Development. Paris: UNESCO, 2021.
100. UNESCO. Education for Sustainable Development Goals: Learning Objectives. Paris: UNESCO, 2017.
101. UNESCO. ICT Competency Standards for Teachers. Paris: UNESCO, 2011.
102. UNESCO. ICT in Education Policy, Infrastructure, and OER. Paris: UNESCO, 2016.
103. UNESCO. ICT in Education: A Critical Literature Review and Its Implications. Paris: UNESCO, 2019.
104. UNESCO. Open Educational Resources: Policy, Costs and Transformation. Paris: UNESCO, 2017.
105. Voogt J., Knezek G. International Handbook of Information Technology in Primary and Secondary Education. Dordrecht: Springer, 2008.
106. Voogt J., Knezek G., Cox M., Knezek D., ten Brummelhuis A. Under which conditions does ICT have a positive effect on teaching and learning? A call to action // Journal of Computer Assisted Learning. 2013. Vol. 29, № 1. С. 4–14.
107. Voogt J., McKenney S. TPACK in teacher education: Are we preparing teachers to use technology for early literacy? // Technology, Pedagogy and Education. 2017. Vol. 26, № 1. С. 69–83.

108. Voogt J., Roblin N.P. A comparative analysis of international frameworks for 21st century competences // Education Research International. 2012. 2012: 1–14.
109. Wieman C.E., Perkins K.K., Adams W.K. Oersted Medal Lecture 2007: Interactive simulations for teaching physics // American Journal of Physics. 2008. Vol. 76, № 4. C. 393–399.
110. Warschauer M., Matuchniak T. New technology and digital worlds: Analyzing evidence of equity in access, use, and outcomes // Review of Research in Education. 2010. Vol. 34, № 1. C. 179–225.
111. Zacharia Z.C., Olympiou G. Physical versus virtual manipulative experimentation in physics learning // Learning and Instruction. 2011. Vol. 21, № 3. C. 317–331.

Електронне видання

Михайло Каленик

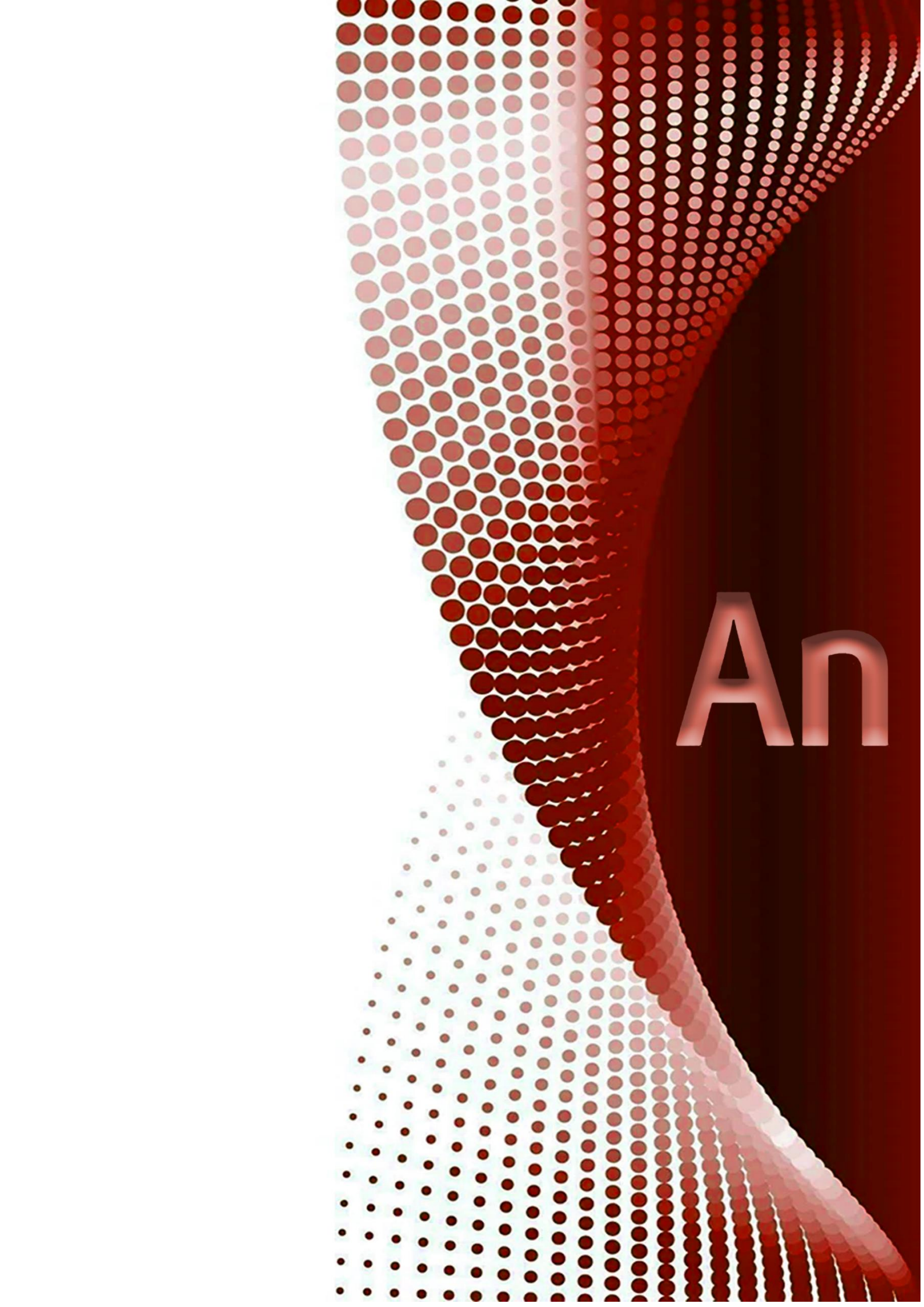
ADOBE ANIMATE У НАВЧАННІ ФІЗИКИ

Навчальний посібник

Підписано у світ 04.03.2026

ФОП Цибульська В.О.
Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції
ДК №6562 від 03.01.2019

ISBN 978-617-8324-74-2


An