Черкаський національний університет імені Богдана хмельницького

Кафедра програмного забезпечення автоматизованих систем

КУРСОВА РОБОТА

З дисципліни «Програмування та алгоритмічні мови»

НА ТЕМУ: «Гра Більярд»

Студента 2 курсу групи КС-211
спеціальності 121 Інженерія
програмного забезпечення
Лєуса В. Д.

Керівник: старший викладач
Гребенович Ю.Є.

Національна шкала:_____
Кількість балів:_____Оцінка:ECTS_____

Члени комісії

_____	_____
(підпис)	(прізвище та ініціали)
_____	_____
(підпис)	(прізвище та ініціали)
_____	_____
(підпис)	(прізвище та ініціали

м. Черкаси – 2022 рік

Зміст

Вступ

Гра більярд – відома всім нам. Вона цікава, та захоплююча, але і не легка, вона потребує певних знань геометрії, елементарної фізики, вправності рук та трішки удачі, що робить її досить популярною і сьогодні, адже кожен хоче спробувати забити більше м’ячів за меншу кількість ударів. Тому актуальність цієї гри сьогодні, важко заперечити, адже багато хто збирається дружньою компанією в пабі чи гральному клубі, щоб добре провести час, за цікавим та розвиваючим заняттям.

Початком розвитку такої гри як більярд, можна назвати XV сторіччя. Перша письмова згадка про нього, знайшли в документах, що стосувалися французького короля Луї XI, що у 1470 році замовив собі більярдного стола. Саме тому і місцем народження більярду вважається – Франція. Є багато версій пояснення чому гра називається саме так, дехто вважає, що французькі слова «bille», що означає «куля», чи «billiard», що означає палиця, і є походженням назви, але можна знайти і згадку англійського дослідника Джона Вілька, що вказував на слова з давньосаксонської мови: «ball» - м’яч та «yerd» - палиця. Початкова версія гри була такою: дві кулі, шість луз, обручі – ворота та вертикальний кілочок, яким не штовхали шари, а били. Зараз же залишили лише кий та кулі. Зелене покриття столу, пояснюється тим, що спочатку в більярд грали на дворі, але потім гру перенесли у закриті приміщення, а покриття стало як згадкою про траву, на якій колись грали в нього.

Метою цієї курсової роботи є підбір правильних алгоритмів та реалізація програмного компоненту гри «Більярд». Гравцеві на початку дано одну білу кулю для удару, основну, та 6 звичайних, що розташовані пірамідкою. Гравців двоє, у кожного по 3 життя, та 5 спроб за першим раундом. Життя віднімаються при потраплянні білої кулі в лузу, спроби зменшуються на одну з кожним наступним раундом, (всього їх 3). Перехід до наступного раунду відбувається тоді, або коли всі 6 куль у лузах, або

один з гравців втрачає всі серця. Ціль гри – забити більше м’ячів за раунд ніж твій суперник та не втратити всіх життів.

Основні задачі роботи:

1. Знайдення алгоритмів для початкової генерації об’єктів, (куль, кию, поля), правильного руху куль у заданому напрямку, перевірки на життя та спроби гравців після раунду та рахунку після закінчення раунду.
2. Побудова блок-схем обраних алгоритмів.
3. Розробка узагальненої схеми для нашої гри.
4. Реалізація програмного продукту гри з усіма умовами.
5. Зробити висновки щодо перспективи використання нашого готового продукту.

Через неспадаючу цікавість людей до цієї гри, «Більярд», її розробка вважається актуальною на даний момент.

Базові поняття, що будуть використані:

1. Stick – кий, наш об’єкт, що задаватиме силу удару по м’ячу та напрямок зміни траєкторії руху основної кулі.
2. Current Player та Second Player, (поточний гравець та другий гравець – відповідно) – два об’єкти, що відповідають за наших гравців, всі алгоритми будуть працювати виключно з поточним гравцем, та лише один мінятиме їх місцями після певних умов закінчення раунду.

Розділ 1. Огляд алгоритмів для реалізації гри «Більярд»

1.1 Алгоритм слідування києм за курсором, для задання напрямку м’яча.

Після того, як розташовано на ігровому полі всі об’єкти, а саме: поле, основний м’яч та кий, що було реалізовано через завантаження картинок формату .png, постає питання, як задавати напрямок, куди вдарити києм, щоб м’яч полетів туди, куди нам потрібно. Шукаючи способи вирішення, було знайдено спосіб, як у деяких ігор 90-х – слідування за курсором миші. Для цього була знайдена стаття, автором якої є Neil Brown [1], де детально описано спосіб задання слідування певного об’єкту за іншим об’єктом по прямій. Вирішення виявилось досить простим, маючи координати точок об’єкта що слідує та об’єкта за яким слідуватимуть, буде використано просту геометрію зі школи, отже, дистанція між цими точками – буде діагоналлю – рівною прямою, і коли об’єкт, за яким слідуватимуть буде перпендикулярно перед об’єктом, що слідує, то тут вектор куди дивитиметься наш кий - не змінюється, вже тільки при переміщенні курсору, ми використаємо прямокутний трикутник, де будемо через тангенс кута, знаходити, на скільки і куди треба повернути об’єкт, що слідує, що вони знову були на прямій лінії, практично це виглядатиме так:

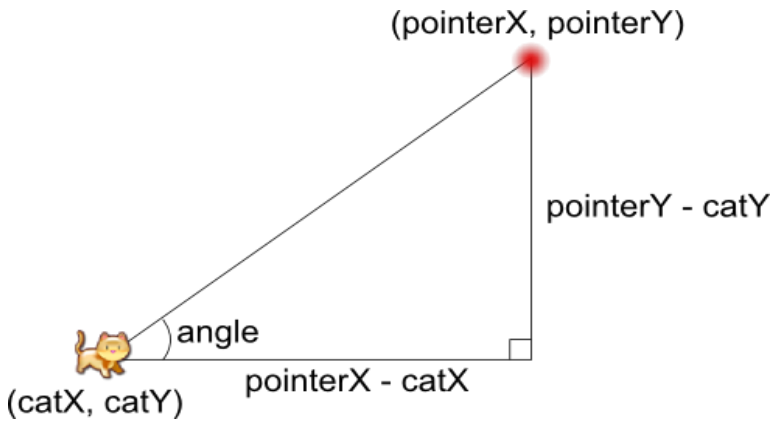


Рис. 1. Приклад визначення кута повороту. [2]

Тепер виводячи це в формули, отримаємо наступне, (α = angle за рис. 1, за pointer – cursor, і cat - stick):

$$\tan(\alpha) = \frac{cursor(Y) - sctick(Y)}{cursor(X) - stick(Y)};$$

Тепер, щоб отримати числове значення, на котре нам повернути зображення stick, ми використовуємо арктангенс – arctan.

$$\alpha = \arctan\left(\frac{cursor(Y) - sctick(Y)}{cursor(X) - stick(Y)}\right).$$

Тобто тепер, знатимемо на скільки повернути наше зображення stick, щоб воно було на одній прямій з курсором, і таким чином ми задаватимемо напрямок удару. Також в вище наведеній статті – описано як це зробити в мові програмування, і вказано, що для коректності використання, а саме, проблема можливого ділення на нуль, рекомендовано використати вбудовану функцію не atan, а atan2, що сама вирішує цю проблему, відмінність, що параметр Y приймається першим, а X – другим.

1.2 Функція передавання сили від stick, до основного м’яча.

Ця задача була вирішена просто. Так як у реалізації гри, наш stick, завжди позиційно прив’язаний до поточної позиції білого шару, реалізовано EventHandler – затискання або натискання лівої кнопки миші, що за певний проміжок часу, що користувач утримує клавішу, збільшує значення сили удару, та потім, передає це значення нашому об’єкту – біла куля, там створюється нова змінна velocity типу Vector, що означатиме нову позицію відносно старої, наскільки пікселів покотиться наш білий шар, і буде постійно виконуватись, щоразу зменшуючись на певну константу, тобто м’яч буде сповільнюватись, проходити меншу кількість пікселів за 1 раз, аж поки не дійде до нуля – повна зупинка м’яча. Тобто, просто переміщуємо м’ячик по координатам, в сторону курсору на певну кількість пікселів за раз, щоразу її зменшуючи, аж до зупинки.

1.3 Алгоритм зіткнення м’яча з ігровими об’єктами, тобто колізії.

Це була найскладніша задача, що повинна була бути вирішена в нашому проекті. Для зіткнень з столом, рішення просте, шукаємо від якої границі наш м’яч відбивається, тоді змінюється знак потрібної нам координати на протилежний, наприклад, якщо від верхньої або нижньої, то буде -Y, якщо від лівого чи правої, то -X. Рішення руху м’ячів після зіткнень, знову було знайдено в старих статтях, автором якої на цей раз є Chad Bercheck [3]. Автор наводить як за 6 дій вирішити нашу задачу, не використовуючи тригонометрії, лише вектори, що працює виключно для круглих об’єктів.

1) Знайти unit normal vector та unit tangent vector. Тобто, знаходимо за формулами нормальний вектор руху при перпендикулярному попаданні м’яча в м’яч, як до перпендикулярних об’єктів, а другий вектор – це одиничний дотичний вектор, що є місцем зіткнення круглих об’єктів в момент зіткнення. Щоб знайти normal vector, віднімемо координати центру другого м’яча від першого:

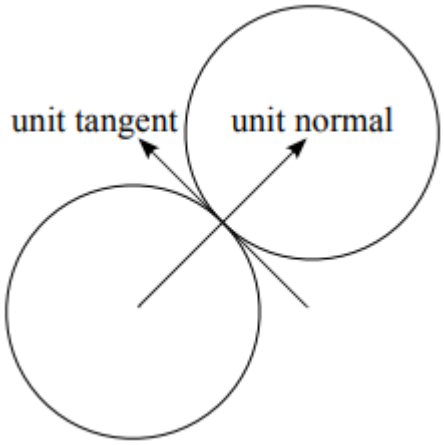


Рис. 2 Приклад нормального та тангенціального векторів. [4]

$$\vec{n} = (x_2 - x_1;\ y_2 - y_1);$$

Тепер знаходиться unit normal vector за наступною формулою:

$$\overrightarrow{un} = \frac{\vec{n}}{\sqrt{n_x^2 + n_y^2}};$$

Тепер unit tangent vector, просто замість значення X, беремо мінусове значення Y від unit normal vector, а за Y – X.

$$\overrightarrow{ut} = (-un_y,\ un_x).$$

2) Тепер потрібно розв'язати вектори швидкості, так, щоб отримати з них нормальний та тангенціальний компоненти. Для цього проектуємо вектори швидкостей на наші unit normal та unit tangent вектори шляхом обчислення скалярного добутку. Нехай v1n буде скаляром (простим числом, а не вектором) швидкість першого об'єкта у нормальному напрямку. Нехай v1t — скалярна швидкість першого об'єкта у тангенціальному напрямку. Подібним чином, нехай v2n і v2t будуть для другого об'єкта. Ці значення знаходяться за допомогою проектування одиничної дотичної нормалі швидкості на unit normal та unit tangent вектори, взявши скалярний добуток. Отримуємо наступні формули:

$$v_{1n} = \overrightarrow{un} \cdot \overrightarrow{v_1}\ ;\ v_{1t} = \overrightarrow{ut} \cdot \overrightarrow{v_1}\ ;\ v_{2n} = \overrightarrow{un} \cdot \overrightarrow{v_2}\ ;\ v_{2t} = \overrightarrow{ut} \cdot \overrightarrow{v_2}\ ;$$

3) Знаходяться нові тангенціальні швидкості, після зіткнень, тут все просто, бо тангенціальні компоненти не змінюються після зіткнень. Єдине, що ми повинні їх позначити, як ті, що після зіткнень.
4) Знаходиться нова нормальну швидкість. Тут використано формули одновимірного зіткнення. Швидкості двох кіл уздовж нормального напрямку перпендикулярні до поверхонь кіл у точці зіткнення, тому це справді одновимірне зіткнення.

$$v'_{1n} = \frac{v_{1n}(m_1 - m_2) + 2m_2 v_{2n}}{m_1 + m_2}\ ;\ v'_{2n} = \frac{v_{2n}(m_2 - m_1) + 2m_1 v_{1n}}{m_1 + m_2}\ ;$$

При розкритті дужок та скороченні це перейде до такого:

$$v'_{1n} = v_{2n};\ v'_{2n} = v_{1n}.$$

5) Конвертується скалярна нормальну та тангенціальна швидкості у вектори, помноживши наш unit normal vector на скалярну нормальну швидкість, отримаємо вектор, що матиме нормальний напрямок і матиме нормальну швидкість. Аналогічно для тангенціальної складової.

$$\overrightarrow{v'_{1n}} = v'_{1n} \cdot \overrightarrow{un};\ \overrightarrow{v'_{1t}} = v'_{1t} \cdot \overrightarrow{ut};\ \overrightarrow{v'_{2n}} = v'_{2n} \cdot \overrightarrow{un};\ \overrightarrow{v'_{2t}} = v'_{2t} \cdot \overrightarrow{ut};$$

6) Знаходиться кінцева векторну швидкість додаючи нормальні та тангенціальні компоненти для двох об’єктів.

$$\overrightarrow{v'_1} = \overrightarrow{v'_{1n}} + \overrightarrow{v'_{1t}};\ \overrightarrow{v'_2} = \overrightarrow{v'_{2n}} + \overrightarrow{v'_{2t}};$$

Саме за цими формулами отримано наші фінальну швидкість та напрямок руху куль після зіткнень.

1.4 Алгоритми перевірки м’ячів у лузах, перевірка та віднімання життів.

Загнання м’ячів у лузу перевіряється простою формулою відстані, тобто якщо дистанція від точки центру кулі до точки центру лузи менша за радіус лузи, то м’яч у лузі, і наш м’яч, видаляється з масиву м’ячів за своїм індексом, а Current Player отримує +1 очко до свого рахунку. Якщо ж гравець заганяє білого м’яча в лузу, то кількість його життів зменшується, а білий м’яч переставляється на своє початкове положення і по ньому б’є наступний гравець.

1.5 Алгоритм перевірки витрачених спроб та їх кількості.

У нас вже написана функція, що робить «постріл» по м’ячу, отже якщо ця функція була викликана, то це означає що поточний гравець вже зробив свою спробу, неважливо чи вона вдала чи ні, тому після виклику цієї функції Current Player втрачає 1 спробу.

1.6 Алгоритм перевірки закінчення ходу та зміни гравця.

Для закінчення ходу гравця, повинні статись певні події: була зроблена спроба і ні один з м’ячів не потрапив до лузи, або ж білий потрапив у неї. Отже якщо якась з цих подій відбулася, то викликається функція, що змінить поточного гравця. Якщо ж поточний гравець загнав м’яча у лузу, то його хід продовжується, за умови, що у нього є спроби. Якщо у одного гравця закінчились спроби, а у іншого ні, і у кожного ще є певна кількість життів, то ходить тільки той гравець, що має спроби, поки не витратить їх, або не змарнує усіх своїх життів.

1.7 Перевірка закінчення раунду та визначення переможця.

Для закінчення раунду потрібно щоб збулась якась з умов: котрийсь гравець втратив всі життя, обидва гравці змарнували всі спроби, всі м’ячі у лузах окрім білого. Перевірка на можливе закінчення раунду робиться після кожного ходу, одного з гравців. Якщо ж хоча б одна умова відбулася, вилітає повідомлення про переможця та дається два вибори, натиснути Esc щоб продовжити гру в наступному раунді, чи натиснути F5 щоб повністю перезапустити гру. Гра перезапускається сама через 2 секунди після завершення раунду лиш чекає вибору гравця, продовжити, чи перезапуститися повністю. Якщо продовжити, то м’ячі та кий повертаються на місця, скидаються результати гравців, Current Player = Player 1 та Second Player = Player 2, але кількість спроб зменшується на 1 в наступному раунді, мінімальна кількість спроб – 3, тоді бажано повністю перезапустити гру через F5, тоді вона коректно почнеться з самого початку, хоча може і сама повернутися до кількості спроб у вигляді 5, після завершення 3 раунду.

Висновок до першого розділу.

Гра відрізняється певними правилами від багатьох варіацій гри «Більярд», але багато чого залишається без змін. Певні задані умови змушують власноруч дописувати певні перевірки, а основні правила гри, знайти певні документації, щодо ігрової механіки руху куль, пострілів та

визначення напрямку як основної кулі, так і інших після зіткнення з кулями чи з столом. Генерація ігрового поля відбувається через відмалювання картинок на певному полі, виставляючи їх послідовно за певними координатами на полі. Поворот кию чи всі колізії відбуватимуться за рахунок функцій, що обраховуватимуть вище наведені формули. А всі перевірки, що описані вище, будуть описані певною функцією з умовами, що за їх виконання даватиме якусь зміну в нашій грі.

Розділ 2. Проектування програмного продукту гри «Більярд»

2.1 Загальна концепція майбутньої гри «Більярд».

Програма матиме реалізацію управління через комп’ютерну мишу, мати графічне зображення поля для взаємодії користувача та гральних об’єктів. Також наш продукт повинен реалізувати всі умови, що були поставлені нам як правила, тобто реалізація нашої гри «Більярд» з усіма потрібними функціями.

Гра повинна містити в собі такі об’єкти:

- Player, що матиме такі властивості: номер, кількість життів, кількість спроб та рахунок за раунд.
- Ігрове поле, що має такі об’єкти: масив м’ячів, кий, та лузи.
- Систему управління києм, тобто його поворот навколо білої кулі та удар по ній і оновлення до наступної позиції білого м’яча.
- Метод перевірки переможця в раунді, та зміни ходу поточного гравця.

2.2 Опис основних алгоритмів гри «Більярд».

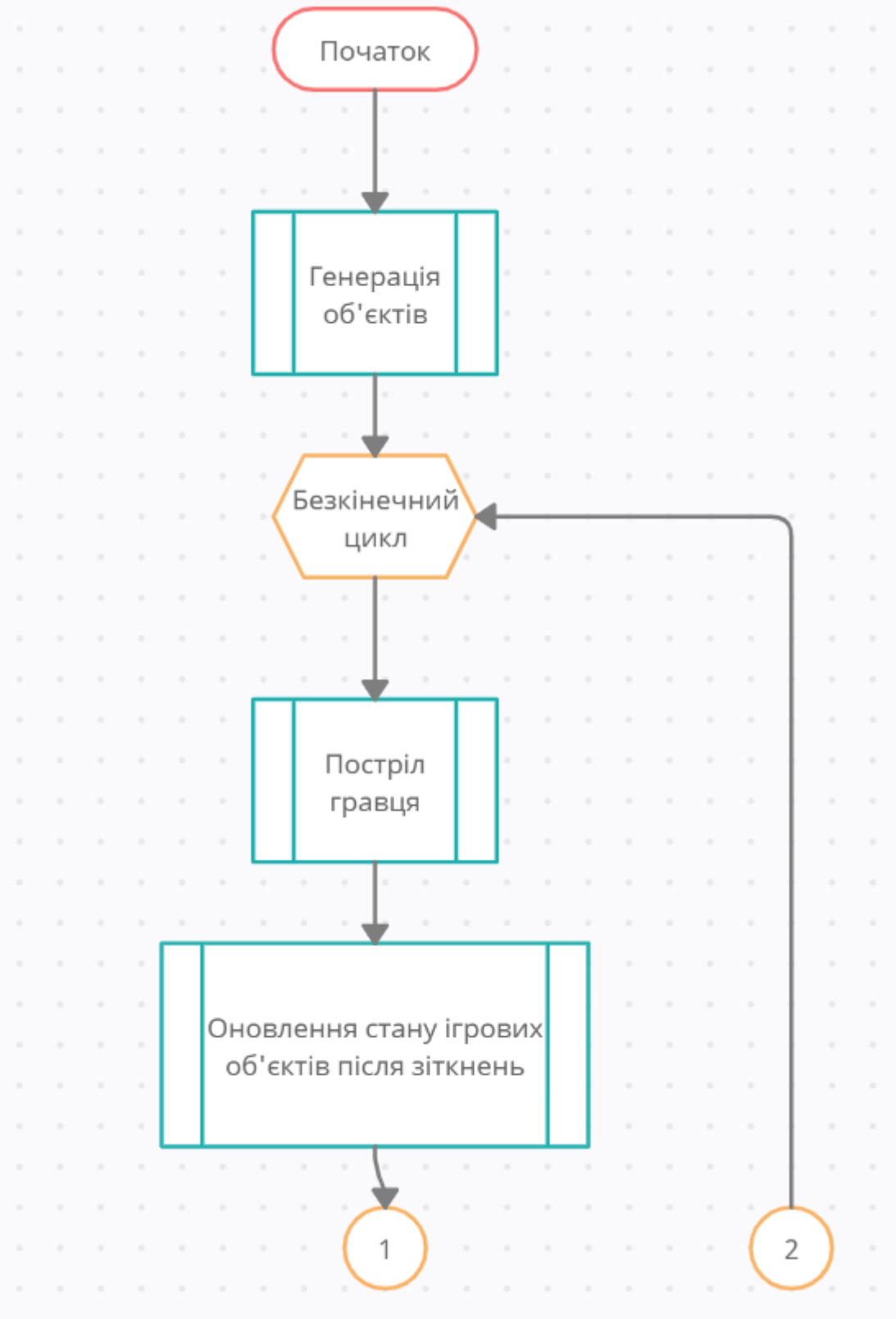
2.2.1 Алгоритм генерації початкових об’єктів.

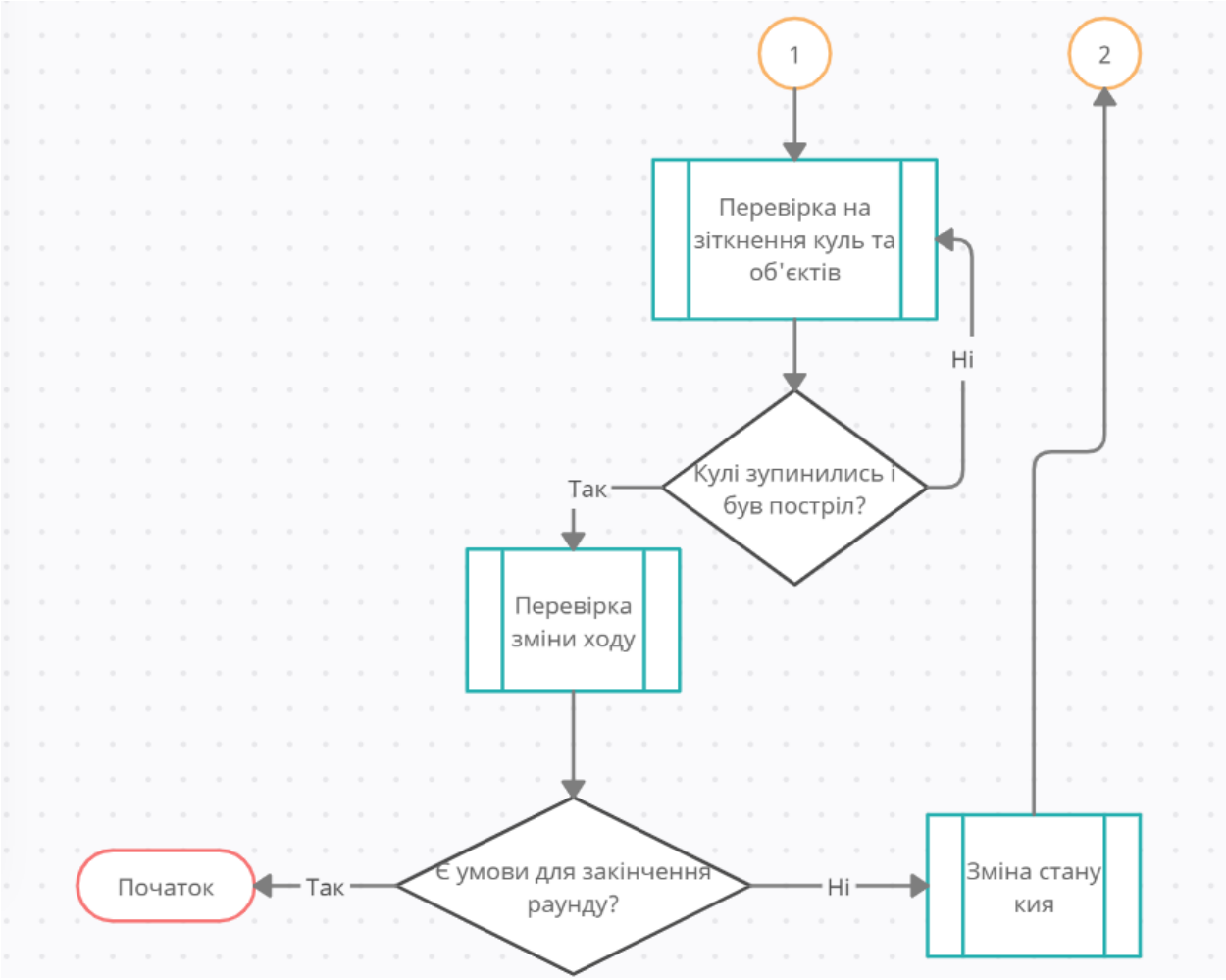
Ігрове поле – це певний canvas, на якому ми малюємо об’єкти: стіл, м’ячі, кий, в реалізації це будуть картинки формату .png, всі вони будуть завантажуватись по черзі, спочатку картинка реального ігрового стола – як задній фон, потім кий, потім всі м’ячі, що будуть представлені одновимірним масивом об’єктів класу Ball, що має такі властивості: позицію та колір, за позицією буде відмальовуватись картинка м’яча, яка в свою чергу обирається за рахунок кольору, введеного нами при створенні об’єкту цього класу.

Кий же буде об’єктом класу Stick, що матиме 2 параметри, початкову позицію, ідентично до позиції білої кулі, та прив’язку до певного м’яча, в нашому випадку – білого.

Всі наступні алгоритми, що виконуватиме наша програма, були описані в Розділі 1. Саме наведені вище рішення і використовуватиме наша програма для реалізації нашого програмного продукту, а саме гри «Більярд».

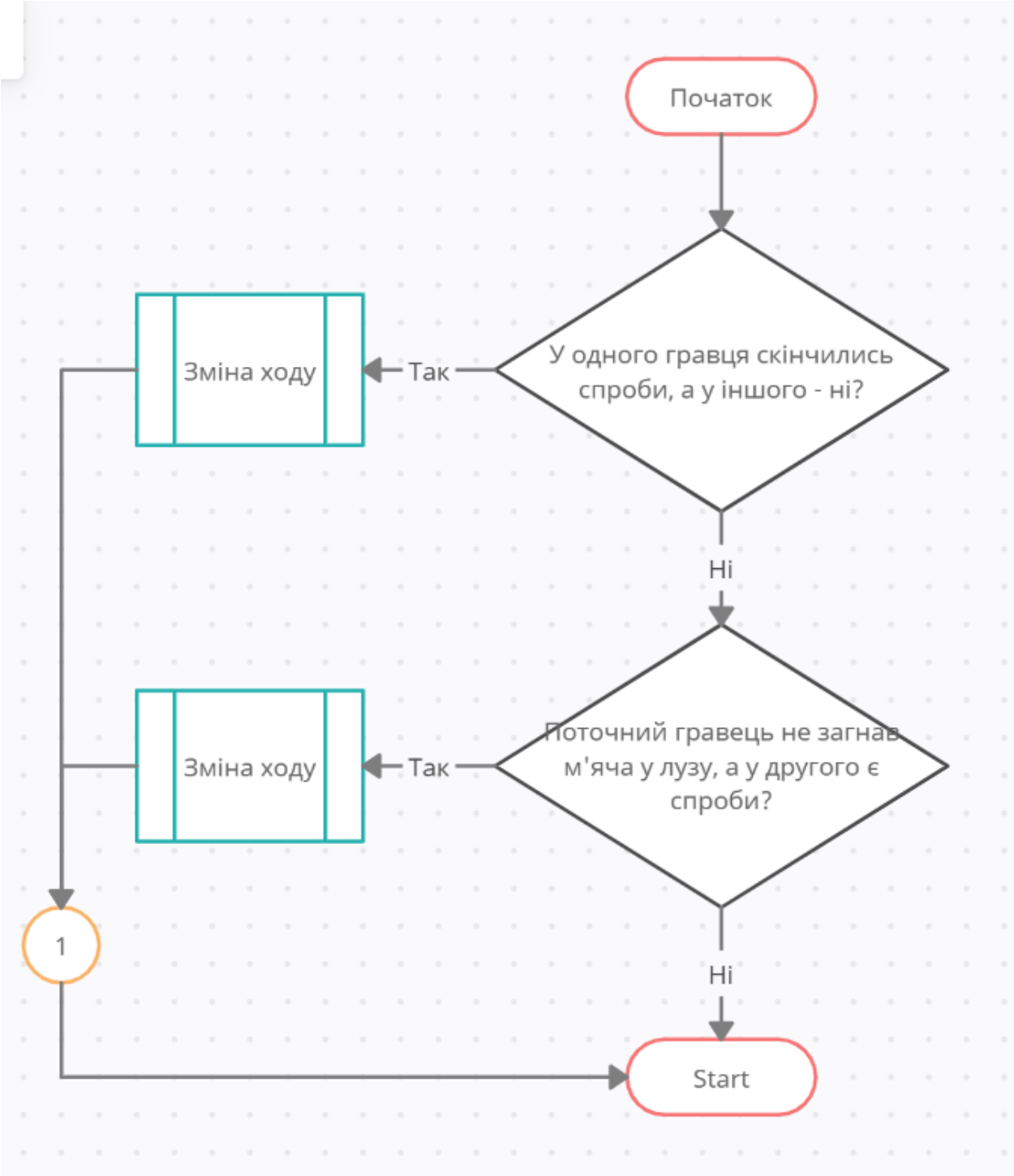
2.3 Блок-схеми для основних алгоритмів гри «Більярд».



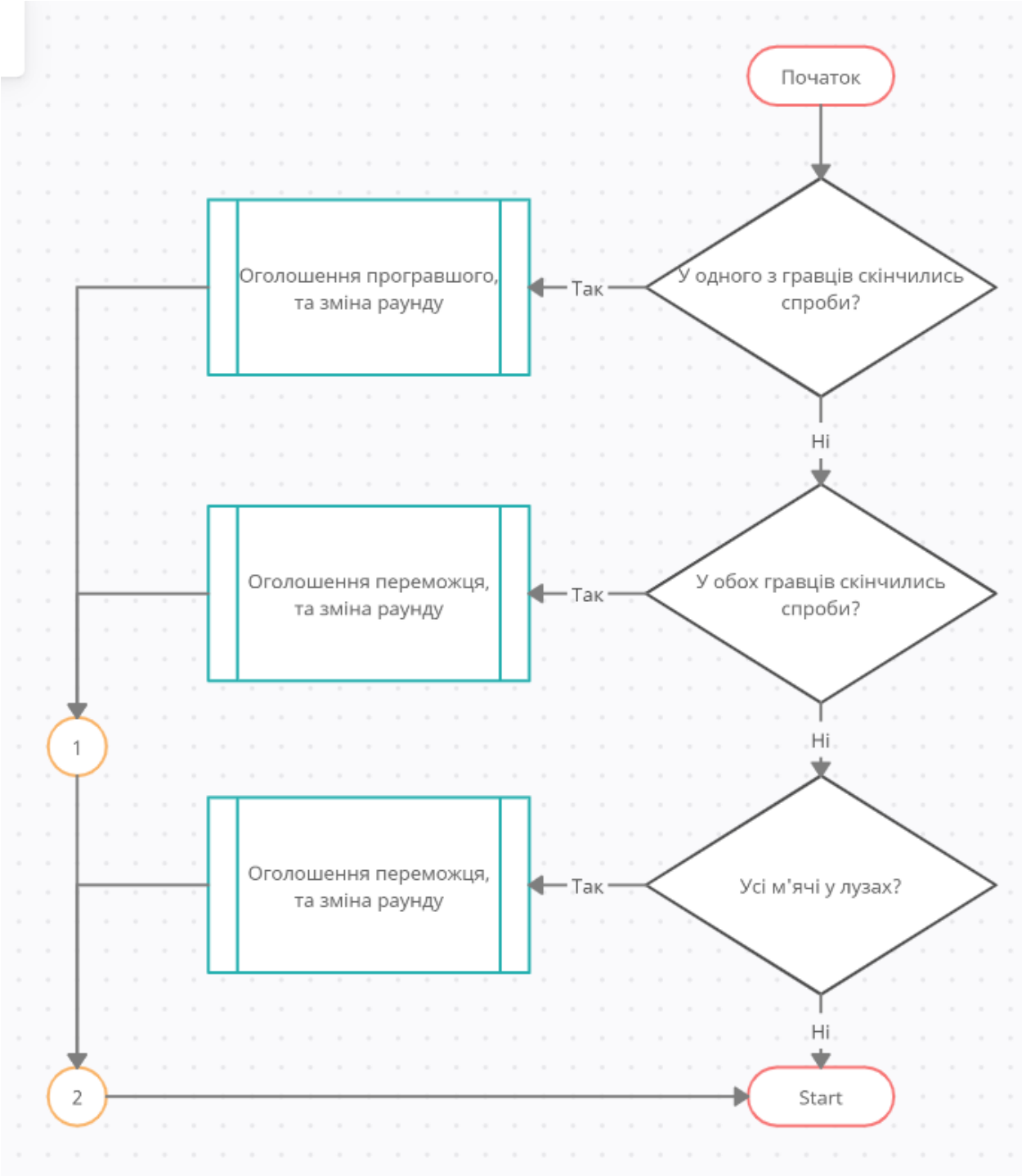


Блок схема основного алгоритму роботи програми.

Гра не завершується, поки гравець чи гравці не закриють її, лише гра переходитиме до наступного раунду, повертаючись на початок, виконання нашої програми. Попередньою блок схемою показано умовну роботу нашої програми. Далі описуються лише перевірки на зміну ходу чи завершення гри, адже всі оновлення станів певних об’єктів, є просто виконанням формул з алгоритмів описаних вище.



Блок-схема операції зміни ходу.



Блок-схема визначення переможця та переходу на наступний рівень.

Висновок до другого розділу.

Опрацювавши всі отримані нам умовою задачі, знайдено необхідні для розробки програмного забезпечення гри «Більярд» алгоритми, що чітко та правильно виконають наші завдання. Тепер подальшою метою є розробка робочого варіанту програми та реалізація її архітектури інструментами обраної мови програмування.

Мета має собою:

- Розробка коду, та кодових абстракцій всіх ігрових об’єктів нашої гри: поле, м’ячі, кий. Та реалізація потрібних для гри алгоритмів, описаних у Розділі 1.
- Розробка правильного керування для взаємодії користувача з програмою.

Розділ 3. Реалізація програмного продукту – гри «Більярд»

3.1 Вибір мови програмування та загальна архітектура програми.

Для розробки нашого програмного продукту обрано мову програмування – JavaScript. Адже вона добре взаємодіє з пристроями введення користувача на сайті, що нам дуже потрібно, також її не сталість полегшує розробку певних частин коду. Також те, що наша гра буде на сайті, дає нам можливість повністю її перезапускати, не закриваючи, натиснувши F5, також завдяки цьому, ми можемо тримати гру як у безкінечному циклі, не пишучи сам цикл, а лиш викликаючи функцію, що є точною входу в гру, в тілі html документу. Тому цією мовою програмування, розроблятиметься логіка програми, та взаємодія користувача з середовищем нашої гри. Почато з логіки та реалізації алгоритмів, наведених у Розділі 1.

3.2 Логіка програмного продукту.

Початковим типом, що створювався, був тип Player. Ми глобально задаємо Current Player та Second Player, для змоги звернення безпосередньо до їх властивостей у будь якій частині коду. Також створено глобальну змінну ROUNDS, що буде реалізувати в програмі, який зараз раунд, та скільки спроб є у обох гравців, та змінну currPlayerScoreTemp, що буде допоміжною для перевірки зміни ходу. JS – не типізована мова програмування, тому ми не можемо явно задати тип, він визначається безпосередньо мовою програмування самостійно, можемо оголошувати змінні лише через let чи const, так як константні змінні не можна змінювати, а нам це потрібно, ми робимо всі вище перечисленні перемінні – типу let.

Програмний код:

```
let currentPlayer = new Player(0, 1);
let secondPlayer =new Player(0, 2);
let ROUNDS = 1;
let currPlayerScoreTemp = currentPlayer.matchScore;
```

```
function Player(matchScore, playerNum){
    this.playerNum = playerNum
    this.playerHealth = 3;
    this.matchScore = matchScore;
    this.attempt = 5;
}
```

Також реалізовано конструктор, що приймає початкове значення рахунку, та номер гравця відповідно.

Далі йде клас Ball – теж один з основних класів, що потрібен для реалізації гри. В ньому зроблено конструктор, що приймає в себе початкову позицію м'яча, та його колір, потім за кольором ми обираємо відповідну йому картинку нашого м'яча, що буде відображена на полі. Також створюємо константи – радіус м'яча та радіус лунки. У цьому класі також реалізовано певні алгоритми з Розділу 1, але про них – пізніше.

Програмний код:

```
const ball_origin = new Vector(25,25)
const ball_diameter = 38
const ball_radius = ball_diameter / 2
const hole_radius = 46

function Ball(position, color){
    this.position = position
    this.velocity = new Vector()
    this.moving = false
    this.sprite = getBallsSpritesByCol(color)
    this.ballColor = color
}
```

Далі йде клас Stick – наш кий. В ньому присутньо багато методів, що реалізують саме: слідкування за курсором, удар, та вимальовка. Ну і звісно конструктор, що приймає у себе початкову позиції, та прив'язку до м'яча, до білого у нашому випадку.

Програмний код:

```
const stick_origin = new Vector(970,11)
const stick_shot_origin = new Vector(950,11)
const max_power = 3500


function Stick(position, onShoot){
    this.position = position
    this.rotation = 0
    this.origin = stick_origin.copy()
```

```
    this.power = 0
    this.onShoot = onShoot
    this.shot = false

}
```

Також є певні константи потрібні для обрахунків.

Тепер останній елемент гри – саме поле. Клас Canvas займається відображенням усіх об’єктів на полі та його очисткою перед початком нового раунду/гри.

Програмний код:

```
function Canvas2D(){
    this.canvas = document.getElementById('screen');
    this.canvasContext = this.canvas.getContext('2d'); // 2d cuz we have a
two-dimensional game
}

Canvas2D.prototype.clear = function ()  {
    this.canvasContext.clearRect(0,0, this.canvas.width, this.canvas.height);
}

Canvas2D.prototype.drawImage =  function (image, position, origin, rotation =
0)  {

    if  (!position){
        position = new Vector()
    }
    if (!origin){
        origin = new Vector()
    }
    this.canvasContext.save()
    this.canvasContext.translate(position.x, position.y)
    this.canvasContext.rotate(rotation)
    this.canvasContext.drawImage(image, -origin.x, -origin.y);
    this.canvasContext.restore()
}

let canvas = new Canvas2D();
```

Далі йде реалізація алгоритмів, описаних у Розділі 1, всі вони працюють як раз за тим самим описом, тому пояснювати наново – немає потреби, лиш вставимо фрагменти коду, та посилання на їх пояснення:

```
Stick.prototype.rotationUpdate = function (){
    let opposite = mouse.position.y - this.position.y
    let adjacent = mouse.position.x - this.position.x

    this.rotation = Math.atan2(opposite, adjacent)
}
```

Функція слідкування за курсором миші києм. [1]

```
Stick.prototype.update = function (){
    if(mouse.left.down) {
```

```
        this.increasingPower()
    }
    else if (this.power > 0){
        this.shoot()
        currentPlayer.attempt--
    }

    this.rotationUpdate()
}

Stick.prototype.increasingPower = function () {
    if (this.power > max_power){
        return
    }
    this.power += 140
    this.origin.x += 5
}

Ball.prototype.update = function (delta){
    this.position.addTo(this.velocity.mult(delta))

    this.velocity = this.velocity.mult(0.984)
    if (this.velocity.calcLength() < 5){
        this.velocity = new Vector()
        this.moving = false
    }
}
```

Методи, що реалізують удар по м’ячу, та заданні йому швидкості в певному напрямку, також реалізують віднімання спроб. [2]

```
 GameWorld.prototype.handleTheCollisions = function () {

    for (let i = 0; i < this.spawnBalls.length; i++){
        this.spawnBalls[i].collideWith(this.gameTable)
        if (this.policy.handleBallInHole(this.spawnBalls[i])) {
            this.spawnBalls.splice(i,1)
        }
        for (let j = i + 1; j < this.spawnBalls.length; j++){
            let firstBall = this.spawnBalls[i]
            let secBall = this.spawnBalls[j]
            firstBall.collideWith(secBall)
        }
    }
}

Ball.prototype.collideWith = function (object) {
    if (object instanceof Ball){
        this.collideWithBalls(object)
    }
    else this.collideWithTable(object)
}
```

```
Ball.prototype.collideWithTable = function (table){
    if (!this.moving){
        return
    }

    let collided = false

    if (this.position.y <= table.topBorderY + ball_radius){
        this.velocity = new Vector(this.velocity.x, -this.velocity.y)
        collided = true
    }
    if (this.position.x >= table.rightBorderX + ball_radius){
        this.velocity = new Vector(-this.velocity.x, this.velocity.y)
        collided = true
    }
    if (this.position.y >= table.bottomBorderY - ball_radius){
        this.velocity = new Vector(this.velocity.x, -this.velocity.y)
        collided = true
    }
    if (this.position.x <= table.leftBorderX + ball_radius){
        this.velocity = new Vector(-this.velocity.x, this.velocity.y)
        collided = true
    }

    if (collided){
        this.velocity = this.velocity.mult(0.984)
    }
}

Ball.prototype.collideWithBalls = function (ball){
    //find a normal vector

    const n = this.position.subtract(ball.position)

    //find distance
    const distance = n.calcLength()

    if (distance > ball_diameter){
        return
    }

    //find minimum translation distance

    const minimumTransDist = n.mult((ball_diameter - distance) / distance)

    //push/pull balls apart

    this.position = this.position.add(minimumTransDist.mult(1/2))
    ball.position = ball.position.subtract(minimumTransDist.mult(1/2))

    //find the unit normal vector

    const un = n.mult(1/n.calcLength())

    // find unit tangent vector
    const ut = new Vector(-un.y, un.x)

    //project velocities onto the unit normal and unit tangent vector

    const v1n = un.dot(this.velocity)
    const v1t = ut.dot(this.velocity)
    const v2n = un.dot(ball.velocity)
    const v2t = ut.dot(ball.velocity)
```

```
    //find new normal velocities

    let v1nTagged = v2n
    let v2nTagged = v1n

    //convert the scalar normal anf the tangential velocities into vectors
    v1nTagged = un.mult(v1nTagged)
    const v1tTagged = ut.mult(v1t)
    v2nTagged = un.mult(v2nTagged)
    const v2tTagged = ut.mult(v2t)

    //update velocities
    this.velocity = v1nTagged.add(v1tTagged)
    ball.velocity = v2nTagged.add(v2tTagged)

    this.moving = true
    ball.moving = true
}
```

Методи що реалізують всі зіткнення. [3]

```
GamePolicy.prototype.isInsideTopLeftHole = function(pos){
    return this.topLeftHolePos.distanceFrom(pos) < hole_radius;
}

GamePolicy.prototype.isInsideTopRightHole = function(pos){
    return this.topRightHolePos.distanceFrom(pos) < hole_radius;
}

GamePolicy.prototype.isInsideBottomLeftHole = function(pos){
    return this.bottomLeftHolePos.distanceFrom(pos) < hole_radius;
}

GamePolicy.prototype.isInsideBottomRightHole = function(pos){
    return this.bottomRightHolePos.distanceFrom(pos) < hole_radius;
}

GamePolicy.prototype.isInsideTopCenterHole = function(pos){
    return this.topCenterHolePos.distanceFrom(pos) < hole_radius;
}

GamePolicy.prototype.isInsideBottomCenterHole = function(pos){
    return this.bottomCenterHolePos.distanceFrom(pos) < hole_radius;
}

GamePolicy.prototype.isInsideHole = function(pos){
    return this.isInsideTopLeftHole(pos) || this.isInsideTopRightHole(pos) ||
        this.isInsideBottomLeftHole(pos) || this.isInsideBottomRightHole(pos)
||
        this.isInsideTopCenterHole(pos) ||
this.isInsideBottomCenterHole(pos);
}

GamePolicy.prototype.handleBallInHole = function (ball) {

    if (this.isInsideHole(ball.position)) {
        switch (ball.ballColor) {
            case 1:
                currentPlayer.matchScore++
                break;
            case 2:
                currentPlayer.matchScore++
                break;
            case 3:
                currentPlayer.matchScore++
```

```
                break
            case 4:
                this.scored = false;
                break
            default:
                break;
        }
        if (!this.scored) {
            ball.position = new Vector(413, 413)
            ball.velocity = new Vector()
            currentPlayer.playerHealth--;
            this.scored = true;
            return false;
        }
        return true;
    }
}
```

Методи, що реалізують загнання куль у лузу та віднімання життів. [4]

```
GameWorld.prototype.checkForSwitch = function (){
    if (currentPlayer.attempt === 0 && secondPlayer.attempt !== 0){
        this.policy.switchTurns()
    } else if (currentPlayer.matchScore === currPlayerScoreTemp &&
secondPlayer.attempt !== 0) {
        this.policy.switchTurns()
        currPlayerScoreTemp = currentPlayer.matchScore;
    } else if (currentPlayer.matchScore > currPlayerScoreTemp) {
        currPlayerScoreTemp = currentPlayer.matchScore
    }
}

GamePolicy.prototype.switchTurns = function () {
    let temp = currentPlayer;
    currentPlayer = secondPlayer;
    secondPlayer = temp;
}
```

Методи, що перевіряють та реалізують зміну ходів. [5]

```
GamePolicy.prototype.getWinner = function () {
    if (currentPlayer.playerHealth === 0){
        swal({
            title: "Round ended",
            text: `Player ${currentPlayer.playerNum} lost all his hearts.
Press Esc to continue or F5 to restart.`,
            icon: "error",
            button: null
        });
        setTimeout(() =>{
            ROUNDS++
            this.reset()
            poolGame.start()
        }, 2000)
    } else if (secondPlayer.playerHealth === 0) {
        swal({
            title: "Round ended",
            text: `Player ${secondPlayer.playerNum} lost all his hearts.
Press Esc to continue or F5 to restart.`,
            icon: "error",
            button: null
        });
        setTimeout(() =>{
            ROUNDS++
            this.reset()
```

```
            poolGame.start()
        }, 2000)
    }

    if (currentPlayer.attempt === 0 && secondPlayer.attempt === 0){
        if (currentPlayer.matchScore > secondPlayer.matchScore){
            swal({
                title: "Round ended",
                text: `Player ${currentPlayer.playerNum} won the game! Press
Esc to continue or F5 to restart.`,
                icon: "success",
                button: null
            });
            setTimeout(() =>{
                ROUNDS++
                this.reset()
                poolGame.start()
            }, 2000)
        } else if (currentPlayer.matchScore < secondPlayer.matchScore) {
            swal({
                title: "Round ended",
                text: `Player ${secondPlayer.playerNum} won the game! Press
Esc to continue or F5 to restart.`,
                icon: "success",
                button: null
            });
            setTimeout(() =>{
                ROUNDS++
                this.reset()
                poolGame.start()
            }, 2000)
        } else {
            swal({
                title: "Round ended",
                text: `DRAW! Press Esc to continue or F5 to restart.`,
                icon: "success",
                button: null
            });
            setTimeout(() =>{
                ROUNDS++
                this.reset()
                poolGame.start()
            }, 2000)
        }
    }

    if (currentPlayer.matchScore === 6 || secondPlayer.matchScore === 6 ||
currentPlayer.matchScore + secondPlayer.matchScore === 6){
        if (currentPlayer.matchScore > secondPlayer.matchScore){
            swal({
                title: "Round ended",
                text: `Player ${currentPlayer.playerNum} won the game! Press
Esc to continue or F5 to restart.`,
                icon: "success",
                button: null
            });
            setTimeout(() =>{
                ROUNDS++
                this.reset()
                poolGame.start()
            }, 2000)
        } else if (currentPlayer.matchScore < secondPlayer.matchScore) {
            swal({
                title: "Round ended",
```

```
                text: `Player ${secondPlayer.playerNum} won the game! Press
Esc to continue or F5 to restart.`,
                icon: "success",
                button: null
            });
            setTimeout(() =>{
                ROUNDS++
                this.reset()
                poolGame.start()
            }, 2000)
        } else {
            swal({
                title: "Round ended",
                text: `DRAW! Press Esc to continue or F5 to restart.`,
                icon: "success",
                button: null
            });
            setTimeout(() =>{
                ROUNDS++
                this.reset()
                poolGame.start()
            }, 2000)
        }
    }
}
```

Метод, що перевіряє витрачені спроби, їх кількість та кількість життів, визначає переможця, та переходить на наступний рівень. [6] [7]

Тепер вставимо скріншоти кінцевої версії нашої гри, з прикладами оголошення переможця/програвшого та зміни раунду за певних умов:

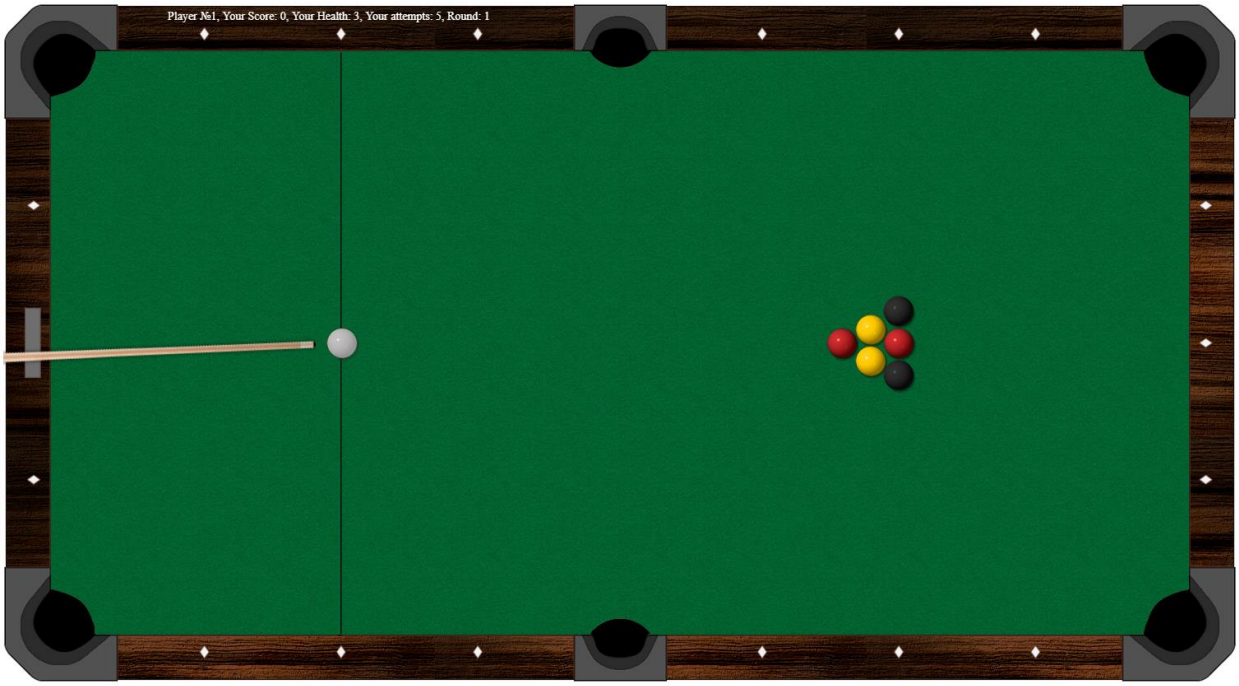


Рис. 3. Кінцевий вигляд початкового поля для гри.

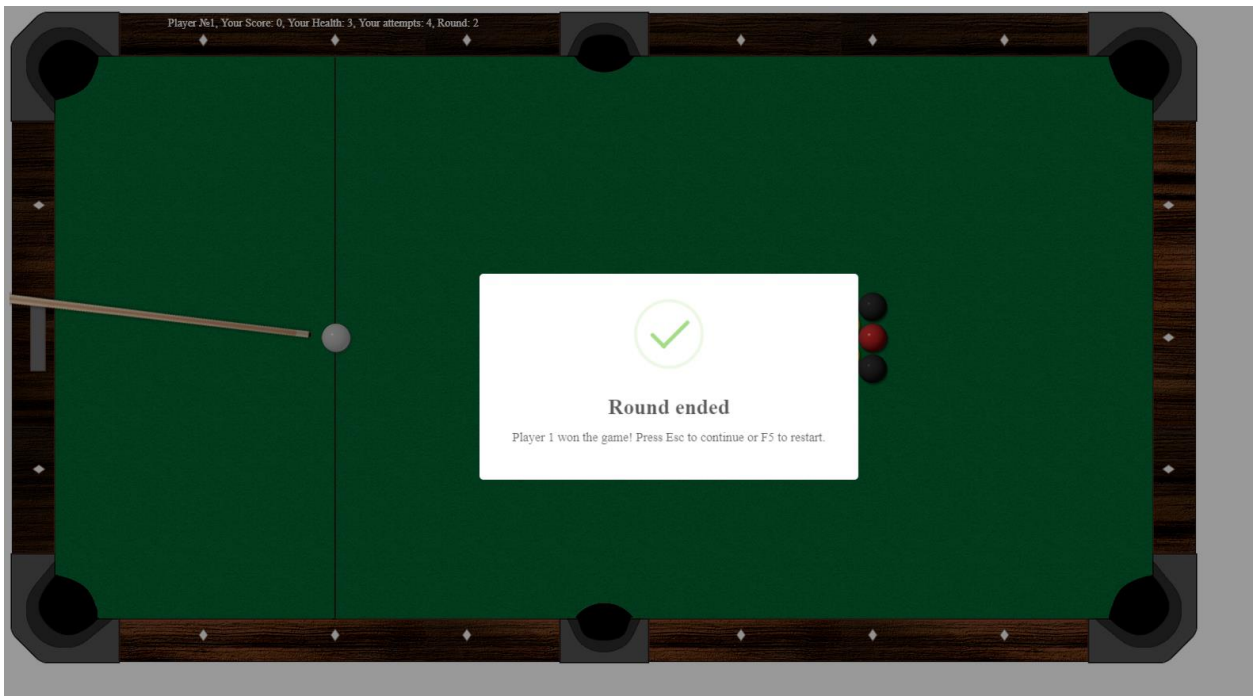


Рис. 4. Оголошення переможця, по закінченню спроб, та перехід на наступний раунд.

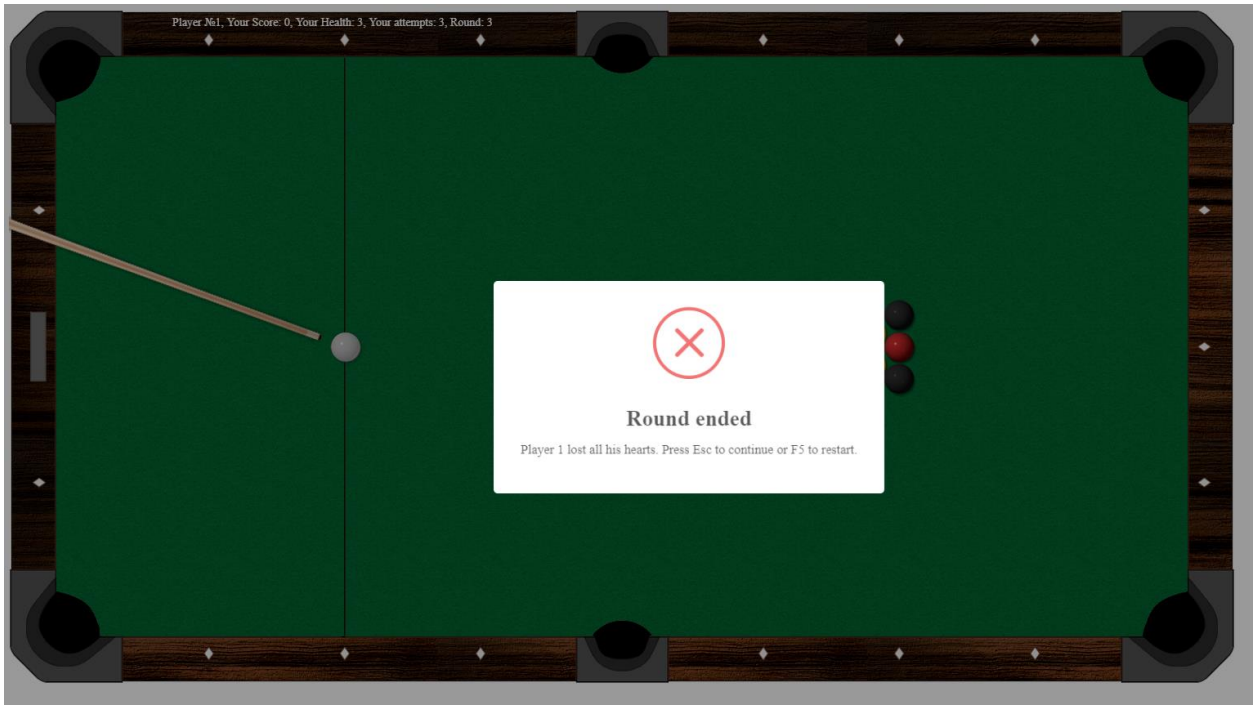


Рис. 5. Оголошення програшного гравця, якщо той втратив всі життя, та перехід на наступний раунд.

Висновок до третього розділу.

В цьому розділі, було обрано мову програмування, для реалізації нашої гри «Більярд», та представлено фрагменти коду, що реалізують

наступний функціонал: слідкування за курсором миші кием, удар по м’ячу, та задання йому швидкості в певному напрямку, віднімання спроб, реалізація всіх зіткнень, загнання куль у лузу та віднімання життів, перевірка та зміна ходу, перевірка витрачених спроб, їх кількісті та кількісті життів, визначання переможця, та перехід на наступний рівень.

Висновок

Для досягнення поставленої мети було виконано певні дії, а саме:

1. Проаналізовано поставлені задачі, та проведено пошук інформації, щодо способів їх вирішення використовуючи статті та інтернет-джерела, що потім допомогло з розробкою нашого програмного продукту.
2. Реалізовано програмну частину гри «Більярд», що має такі можливості:
    a. Автоматичну генерацію початкових об’єктів та поля для гри.
    b. Спосіб взаємодії користувача з нашим програмним продуктом.
    c. Обрахування ходів, життів, загнаних у лузу м’ячів, визначення переможця раунду, та перехід на наступний.
    d. Обрахування правильних руху та швидкості м’ячів.

Для кінцевої реалізації було обрано мову програмування JavaScript, через її переваги у взаємодію з пристроями введення користувача, та не строгою типізованістю мови, що дало більшу гнучкість для розробки гри «Більярд».

Найдоцільнішою сферою використання продукту, є розваги та відпочинок з друзями чи сім’єю, адже ця гра підходить людям будь якого віку. Можна влаштовувати турніри чи змагання між друзями чи сім’єю, щоб мати більший стимул правильно попадати по кулям та загнати їх всі у лузу.

Покращити даний програмний продукт можна шляхом додання: меню, з різними пунктами, наприклад таблиця рекордсменів, розробка якіснішого графічного оформлення гри, та добавлення штучного інтелекту, чи так званого «бота», для гри з ним, коли ви самі. Також можливо реалізувати під більшу кількість девайсів та операційних систем окрім Windows та стаціонарного комп’ютеру.

Список літератури та інтернет джерел

1. Більярд - Вікіпедія [Електронний ресурс] // Wikimedia project – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%91%D1%96%D0%BB%D1%8C%D1%8F%D1%80%D0%B4.
2. Berchek C. 2-Dimensional Elastic Collisions without Trigonometry [Електронний ресурс] / Chad Berchek // Vobarian Software. – 2009. – Режим доступу до ресурсу: https://www.vobarian.com/collisions/2dcollisions2.pdf.
3. Brown N. They’ve got atan, You Want atan2 [Електронний ресурс] / Neil Brown // The Sinepost. – 2012. – Режим доступу до ресурсу: https://sinepost.wordpress.com/2012/02/16/theyve-got-atan-you-want-atan2/.
4. JavaScript | MDN [Електронний ресурс] // MDN Web Docs – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/JavaScript.