Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут ім. Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Курсова робота

з дисципліни «Інженерія програмного забезпечення»

на тему: «Інтерактивна дошка для гри “Нарди”»

Виконав:
студент 2 курсу ФІОТ,
група ІО-21
Безщасний Роман Русланович
Керівник:
Старший викладач кафедри ОТ
Васильєва М.Д.

Київ – 2024

Анотація

Об’єктом розробки виступає програмний додаток з інтерфейсом інтерактивної дошки для гри «Нарди».

Ключові слова: інженерія програмного забезпечення, графічний інтерфейс, інтерактивна дошка, поле переходу, стек фішок гравця, варіації ходу.

Метою даної курсової роботи є покращення роботи з ООП, програмним забезпеченням, графічним інтерфейсом, набуття нових або покращення старих навичок у створенні та веденні проекту, предмету «Інженерія програмного забезпечення», а також засвоєння методів SOLID.

Як методи досліджень я використав написання суміжні з точки зору розробки графічного інтерфейсу програми, завдяки онлайн ресурсам, також аналізував вже наявні проекти з відкритим кодом, що допомогли мені в опануванні використання нових інструментів для роботи з інтерфейсом програми, створеної за допомогою Python коду та допоміжним кодом мови kivy language, що є деяким розширенням модулю Kivy, останнім з цього переліку буде пошук та опрацювання літератури, що допомогла розібратися зі структурою проекту, а саме – використаним шаблоном проектування та зв’язком між об’єктами різних класів.

Результатами створення цієї гри, стали досить цікавий досвід та дуже непогана гра – цікава, відносно проста, трішки азартна, тому що враховує елемент удачі, а також різноманітна у своїх варіаціях позицій. Гра працює безперебійно та досить швидко, враховуючи, що зазвичай мова програмування Python не призначена для створення ігор.

Кількість сторінок: 28; ілюстрацій: 17; додатків: 2; джерел: 7

Зміст

Словник

Програмне забезпечення (ПЗ) - це сукупність програм, призначених для забезпечення роботи інформаційної системи. Воно призначене для управління складовими системи, автоматизації процесу, обробкою даних та забезпеченням інтерфейсу для взаємодії між персональним комп’ютером та користувачем.

Ігрова логіка – сукупність правил та патернів, які є обов’язковими або опціональними до використання під час гри та які враховуючи всі можливі вийнятки та комбінації для будь-якого положення фішок або фігур гравців.

Види переміщення: зайняття пустого поля, збільшення стеку вже зайнятого поля, пропуск ходу(частковий, повний), перехід до початкового стеку, завершення гри.

Комбінації випадіння числа для ходу на кубиках: звичайне число, що означає 2 можливих ходи для 2 різних фішок або однієї; дубль – 4 однакових можливих ходи для будь-яких фішок.

Вступ

Проблемою, що розкриває та пропонує вирішення дана курсова робота, є розповсюдження цієї чудової гри, яка пов’язує мене з моїм дитинством, коли я грав зі своєї бабусею та дідусем. Для мене це досить важливі спогади та класні відчуття після зіграної партії, тому я б хотів поділитися ними з іншими, незважаючи на те, що через нестачу досвіду та знань, вигляд моєї гри може бути не дуже сучасним та «ефектним» до використання, але функціонал є повним, гра виглядає достойно, тому я вважаю, що своєї цілі я досягнув.

Мета даної курсової роботи була обґрунтована в анотаціях, але якщо казати про необхідність використання комп’ютерних засобів, то найбільш резонною причиною буде легкість поширення за допомогою розповсюдження додатку, що містить у собі код та функціонал моєї програми. Так як сьогодні ми знаходимося у світі, який швидко розвивається в технологічному плані – більшість бажаючих у щось пограти перше про що подумають, коли захочуть зіграти – це завантажити гру з Інтернету або якогось спеціального додатку-маркету для сертифікованих програм, тому є необхідність підлаштовуватися під потреби користувача та робити свої продукти більш доступними для нього, через що варто писати ігри для комп’ютеру та смартфонів.

1. Огляд MVC, постановка задачі

1.1 Огляд MVC

MVC (Model-View-Controller) - це архітектурний шаблон проектування, який використовується для розробки програмного забезпечення з графічним інтерфейсом користувача. Цей шаблон допомагає відокремити логіку додатка, його представлення та спосіб взаємодії з користувачем, щоб код став більш читабельним, масштабованим та підтримуваним.

Він розділяється на 3 взаємопов’язані частини:

Модель:

- Серцевина програми, відповідальна за зберігання та обробку даних.
- Отримує команди від контролера (або, іноді, безпосередньо від вигляду).
- Виконує необхідні операції з даними та повертає результат.

Вигляд (інтерфейс):

- Забезпечує взаємодію з користувачем.
- Відповідає за відображення даних, отриманих від моделі.
- Приймає вхідні дані від користувача.

Контролер:

- Посередник між моделлю та виглядом.
- Отримує вхідні дані від користувача через вигляд.
- Інтерпретує ці дані та надсилає команди моделі для оновлення.
- Оновлює вигляд новими даними, отриманими від моделі.

1.2 Постановка задачі

1.2.1 Загальне завдання

Створити повноцінну гру нарди для двох гравців, зробити інтерфейс простим, зрозумілим та зручним в користуванні, не обтягуючи занадто великою кількістю деталей.

1.2.2 Функціональні вимоги

Інтерфейс користувача:

1. Можливість вибору сторони(кольору фішок).
2. Наявність індикаторів ходу.
3. Наявність табличок з рахунками перемог та поразок гравців.
4. Місце з надписом для надання додаткових інструкцій для гравця, в разі спірної ситуації.

Функціональність гри:

1. Здатність переміщення фішок по дошці.
2. Здатність утворювати стек з декількох фішок одного кольору.
3. Перевірка ходів, які не відповідають правилам.
4. Кнопки або інші об’єкти для взаємодії з користувачем та виправлення помилок, які виникають в процесі гри.
5. Автоматичне виявлення та сповіщення про кінець гри, а також надсилання повідомлення про перемогу та кінець гравцям.
6. Можливість перезавантаження гри зі збереженням результатів попередніх раундів.
7. Рахунок фішок у фінальному етапі раунду та рахунок перемог кожного гравця.

1.2.2 Інструменти розробки

Програма спроектована та реалізована за допомогою мови програмування Python, а також її модулю Kivy та додатковому інструменту цього модулю – kivy language з використанням базових структур ООП та патернів Composite і Observer.

2. Проектування програмного забезпечення

2.1 Опис додатку

Програма передбачає один тип користувача в 2 екземплярах – перший гравець(білі фішки) та другий гравець(чорні фішки). Перша дія, яку має зробити завжди тільки перший користувач – це кинути кубики, після чого гра починається.

Гравці можуть переміщувати фішки, накладати їх одна на одну, перекидати кубики, скидати номер обраної фішки, щоб змінити свій вибір, пропускати ходи

та завершувати процес гри завдяки фінальному етапу. Також перезапускати гру зі збереженням результатів попередніх раундів.

2.2 Прецеденти користування

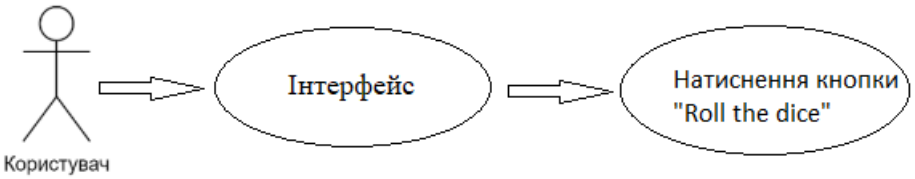

Рисунок 2.1 Прокрутка кубика для ходу гравця(користувача)

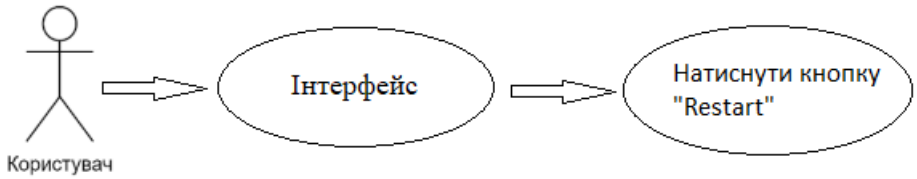

Рисунок 2.2 Перезапуск гри

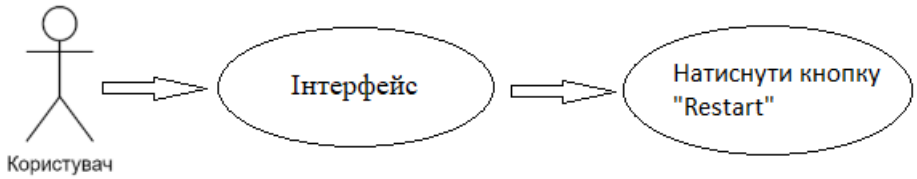

Рисунок 2.3 Перезапуск гри

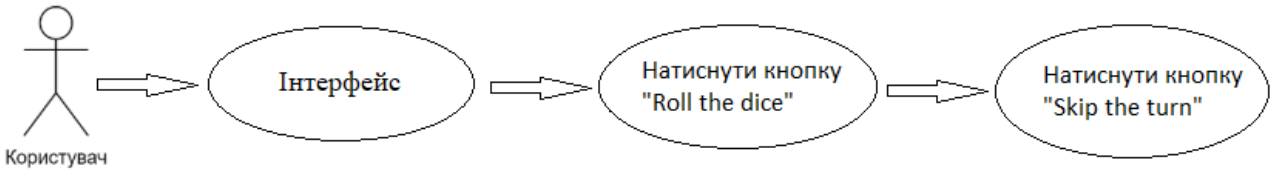

Рисунок 2.4 Пропуск ходу

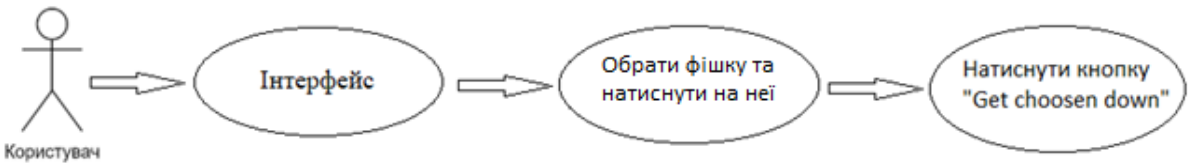

Рисунок 2.5 Налаштування щоб змінити вибір після

натискання на конкретну фішку

Рисунок 2.6 Зробити хід

Рисунок 2.7 Зробити хід у фінальному етапі гри

3. Розробка програмного додатку

3.1 Опис класів

Клас Checker – є моделлю віджета фішки, з якою відбуваються зміни розташування та всі інші махінації за допомогою зміни кольору, тому що спочатку всі фішки вже розставлені на кожному з 24 можливих місць на полі, проте на початку окрім тих фішок, що є знаходяться на початкових позиціях, всі інші пофарбовані в колір фону, що дає їм характеристику в програмі як «невизначені».

Клас GamePart – є моделлю лейаута BoxLayout, а також є однією з 24 рівних частин, що розподілені по всьому полю та містять у собі фішку певного кольору та з певною кількістю (точніше просто надписом Label біля фішки). В цьому класі присутній метод або якщо точніше @staticmethod on_size(), що дає змогу змінювати розміри самої площини відносно розміру самого вікна.

Клас GamePlace – є контролером для регуляції ігрового простору та ігрових ситуацій, він включає до себе великий @staticmethod on_touch_down(), що відповідає за реакцію програми на дотики або натиснення кнопки миші на площині ігрового поля. В цьому методі прописана процедура перевірка всіх можливих ігрових ситуацій для оптимізації програми, тому що можливості додати ще методи для спрощення роботи з додатком та програмою не

знайшлося, тому я вирішив все це залишити в одному методі. Також цей об’єкт є свого роду контейнером для отримання додаткових інструкцій ззовні завдяки змінені значень змінних, які на момент створення цього об’єкту мали значення None. Зроблено це для взаємодії з об’єктом класу PlayerPlace через змінення аргументу під час створення класу MainLayout.

Клас GamePlace – є інтерфейсом та водночас контролером для взаємодії користувачів з логікою гри за допомогою натискання кнопок з різними функціями, але також для відображення кількості фішок, що зникають у фінальному етапі гри, а також для відображення кількості перемог кожного гравця. Також, для оптимізації коду цей клас був в більшій частині створений та сформований за допомогою додаткового інструменту в модулі Python Kivy – kivy language, що уможливлює спрощення написання графічної частини коду та її оптимізації за допомогою вбудованих інструкцій цієї мови.

Клас MainLayout – є головним контролером програми, тому що є містить життєво необхідні методи giving_the_dices(), turning_down_the_checker(), skip_the_turn(), а також створює самі об’єкту класів GamePlace та PlayerPlace, і перенаправляє необхідні дані до змінних GamePlace за допомогою їх видобуття з PlayerPlace, що і уможливлює взаємодію цих двох різних об’єктів. Також, «причіпляє» до кожної кнопки PlayerPlace необхідний метод для нормального функціонування цієї самої кнопки та її функціоналу.

Останній клас MyApp – є головним інтерфейсом програми, що створює об’єкт MainLayout та використовує всі додаткові налаштування для побудови ігрового простору, а робить він все це за допомогою @staticmethod build().

3.2 Обґрунтування засобів реалізації

Програму написано за допомогою однієї з найпоширеніших мов програмування Python. Перед початком написання курсової роботи я бачив 2 способи реалізувати цей проект: за допомогою Python або Java. Так як я більше знайомий з мовою програмування Python, а також я хотів покращити свої навички та знання ООП в Python та вивчити нові можливості цієї мови для роботи з графікою, то й обрав саме його.

Одним з найшвидших, а також одного з найбільш поширених модулів для взаємодії з графічним інтерфейсом в Python є модуль kivy. Також я неодноразово згадував в поясненнях вище про додатковий інструмент взаємодії з цим модулем – kivy language, який пишеться в окремому файлі, але чудово взаємодіє з Python кодом.

4. Тестування програмного забезпечення

Завдяки неодноразовому тестуванню я мав можливість виправити помилки, допущені при поступовому написанні програми, а також зараз завдяки ще одному тесту зможе через screenshot екрану зможу передати всі необхідні приклади взаємодії з програмою.

При запуску ми побачимо, що програма автоматично розширюється відносно габаритів екрану на всю площину, що зроблено для комфорту користувача.

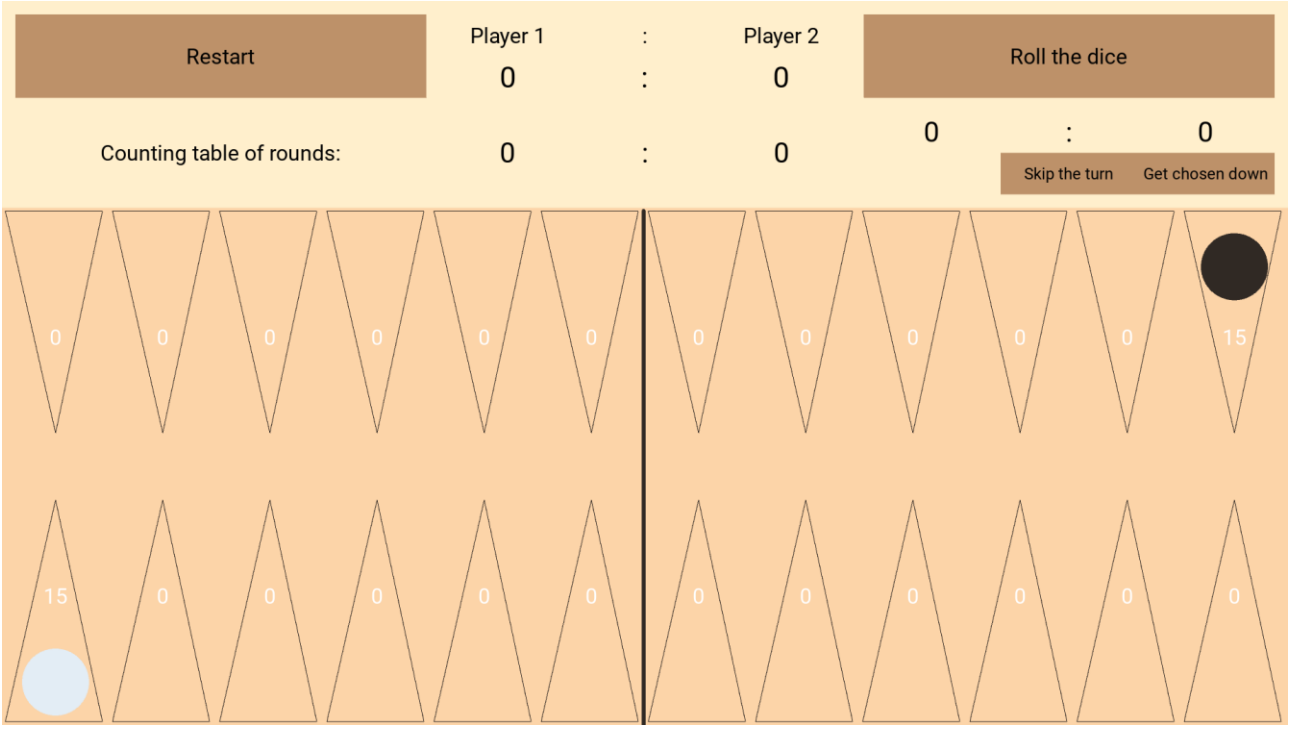

Рис. 4.1 Початковий екран

Ми бачимо 2 частини ігрового поля, з двома видами фішок – білі та чорні, що розташовані навпроти один одного, а також кількість фішок на початку гри на одному місці складає 15 з кожної сторони. Всі значення раундів, результатів гравців, як і чисел, що випали на кубиках дорівнюють 0, тому щоб почати натискаємо кнопку «Roll the dice» та робимо перші ходи.

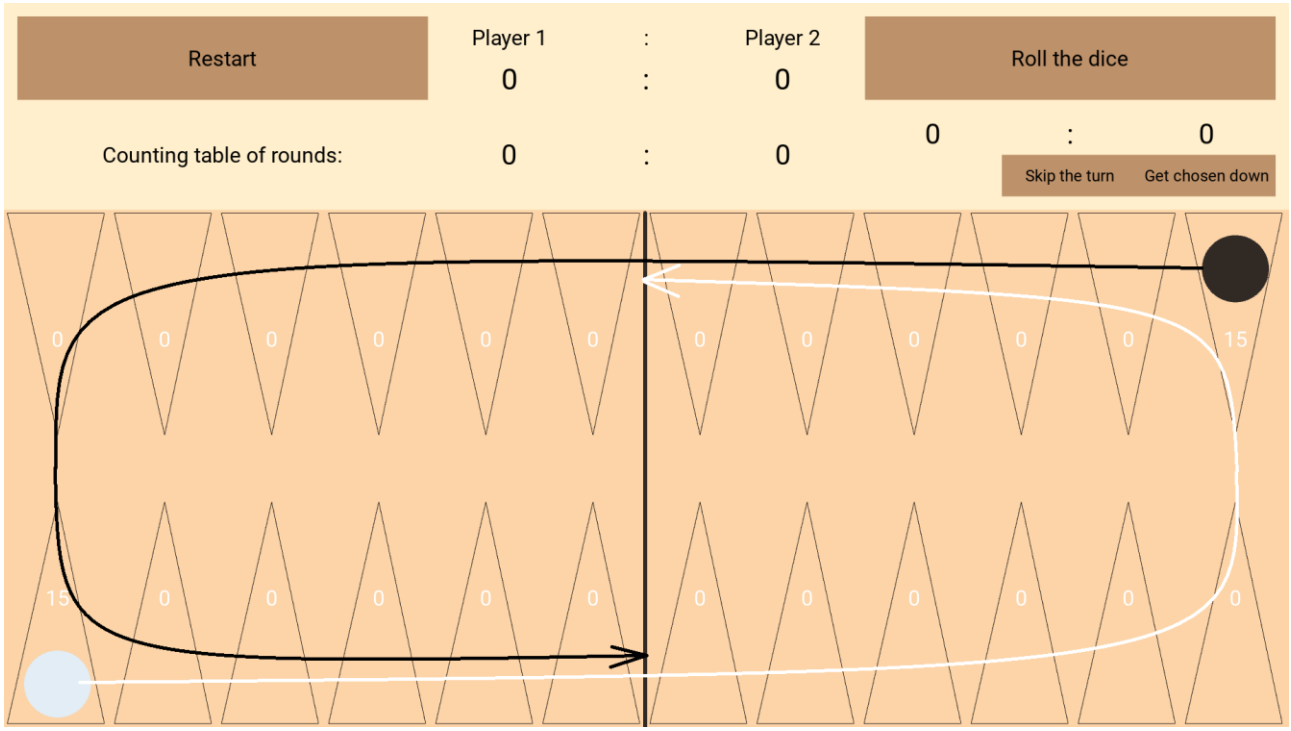

Рис. 4.2 Напрямок ходу фішок та розташування цілі

В нардах фішки ходять проти годинникової стрілки та, починаючи з першої чверті доходять до останньої, і коли всі фішки одного кольору знаходяться в межах останньої чверті – в гравця, керуючого цими фішками з’являється можливість перейти до фінального етапу гри – «Повернення до стеку».

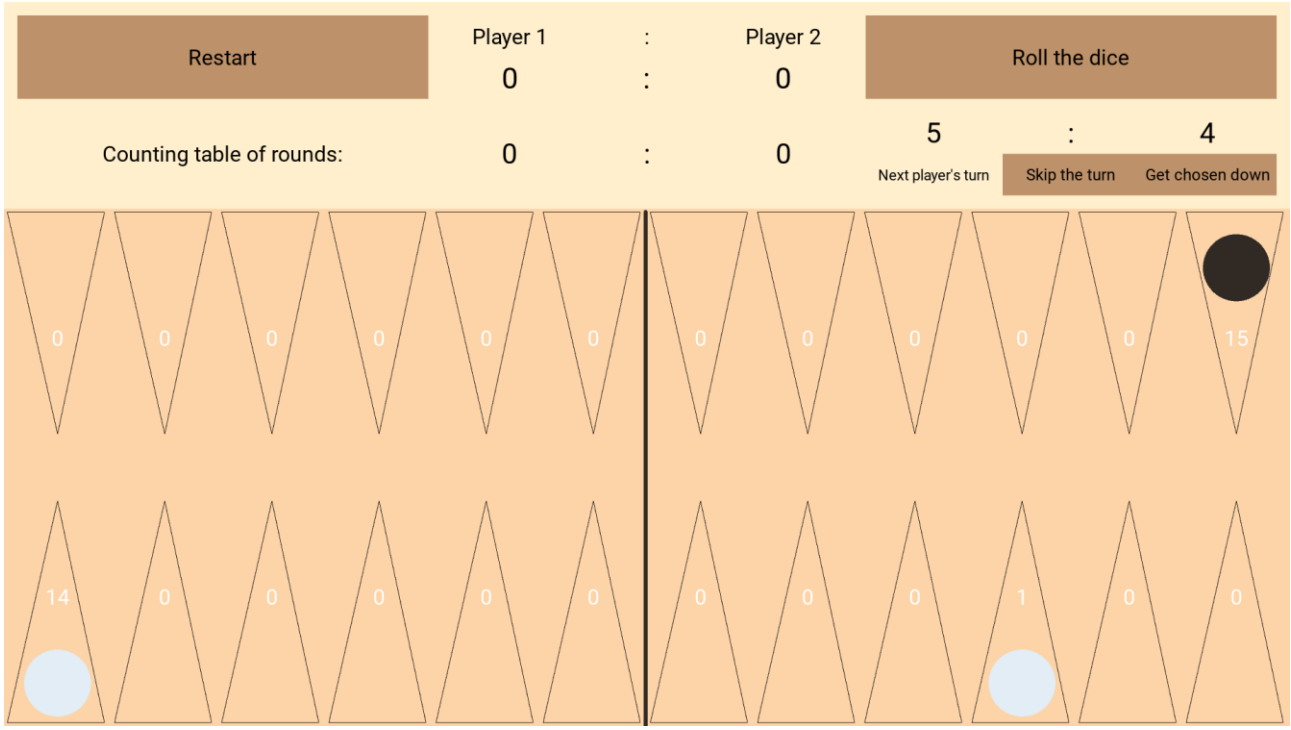

Рис. 4.3 Перший хід

Гравець з білими фішками кинув кубики, на яких випали числа 5 та 4, після чого перший гравець використав обидва ходи на одну фішку. Також по закінченню ходу ми можемо побачити надпис біля чисел, що випали на кубику: «Next player’s turn», що символізує про закінчення ходу та перехід до наступного етапу.

Важливо!!!

Для наступних тестів та зображень я зменшу кількість фішок до 3 для обох гравців для пришвидшення процесу тестування та відображення в більш зручному вигляді механіки гри та її закінчення.

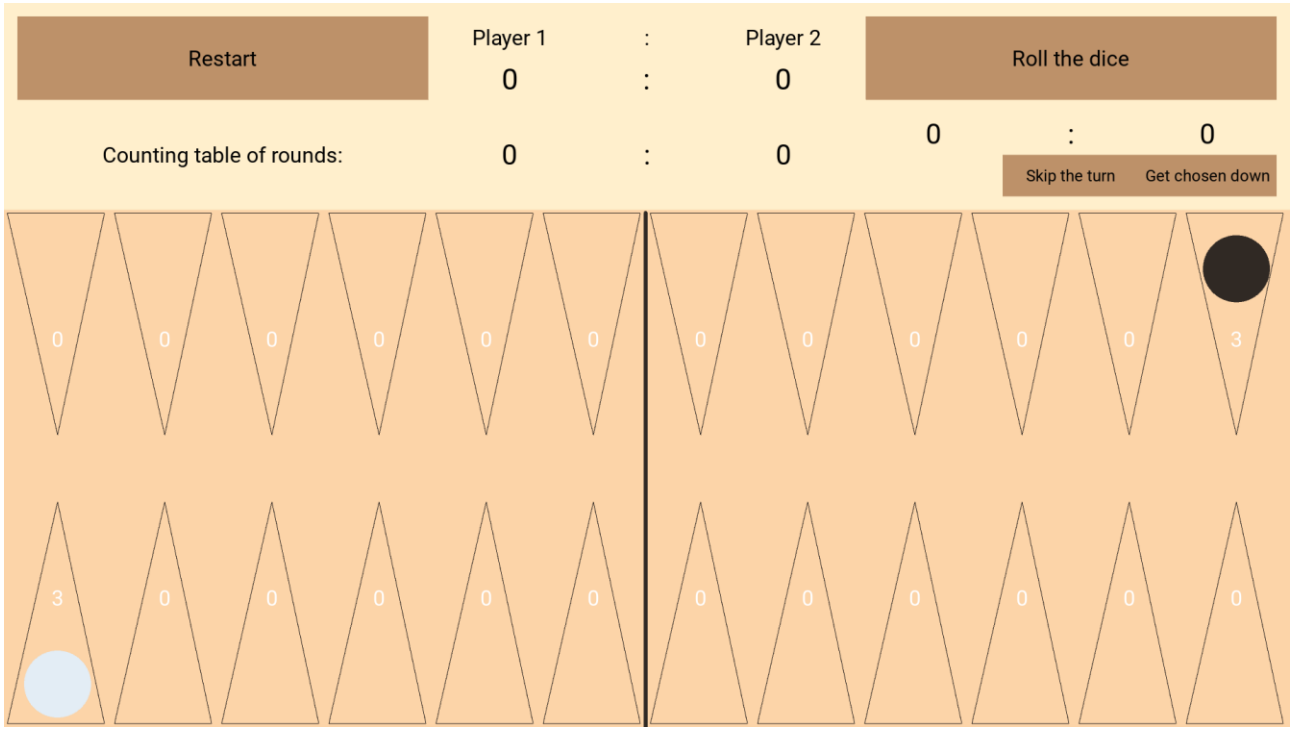

Рис. 4.4 Спрощений варіант гри для тестування

Тепер переходимо до цікавих ігрових моментів, хоча більшість з них можна буде показати тільки при прямому користуванні:

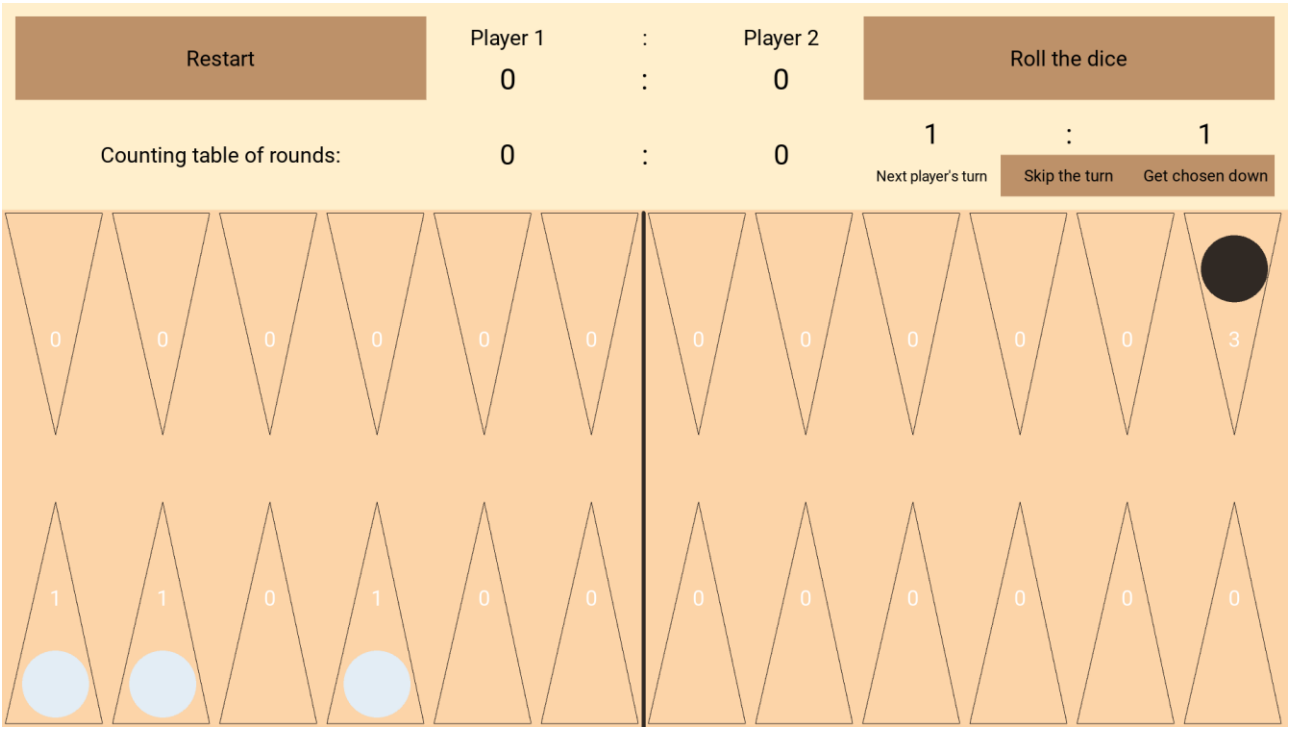

Рис. 4.5 Демонстрація ходу при дублю

Дубль: 4 однакових ходи, з однаковою механікою, що й звичайний хід(4 спроби ходу можна використати для різних фішок), але також при дублі дозволяється зняти 2 фішки з початкового поля одночасно, хоча це тільки одна з варіацій гри, тому в моїй версії 2 фішки можна знімати навіть без дублю.

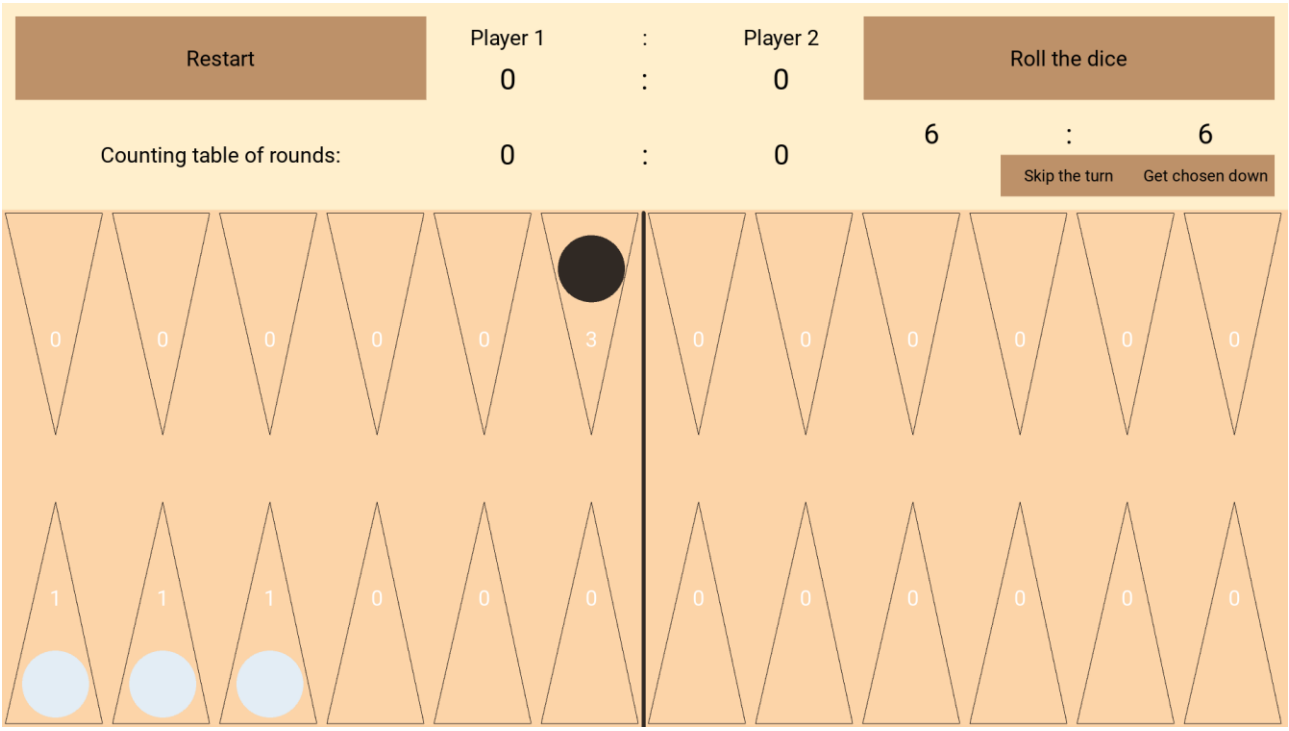

Рис. 4.6 Демонстрація «неможливого ходу»

На рисунку можемо побачити, що гравець «чорних» опинився в стані, коли не може закінчити свій хід звичайним способом, в такому випадку він натискає кнопку «Skip the turn» та хід переходить до наступного гравця.

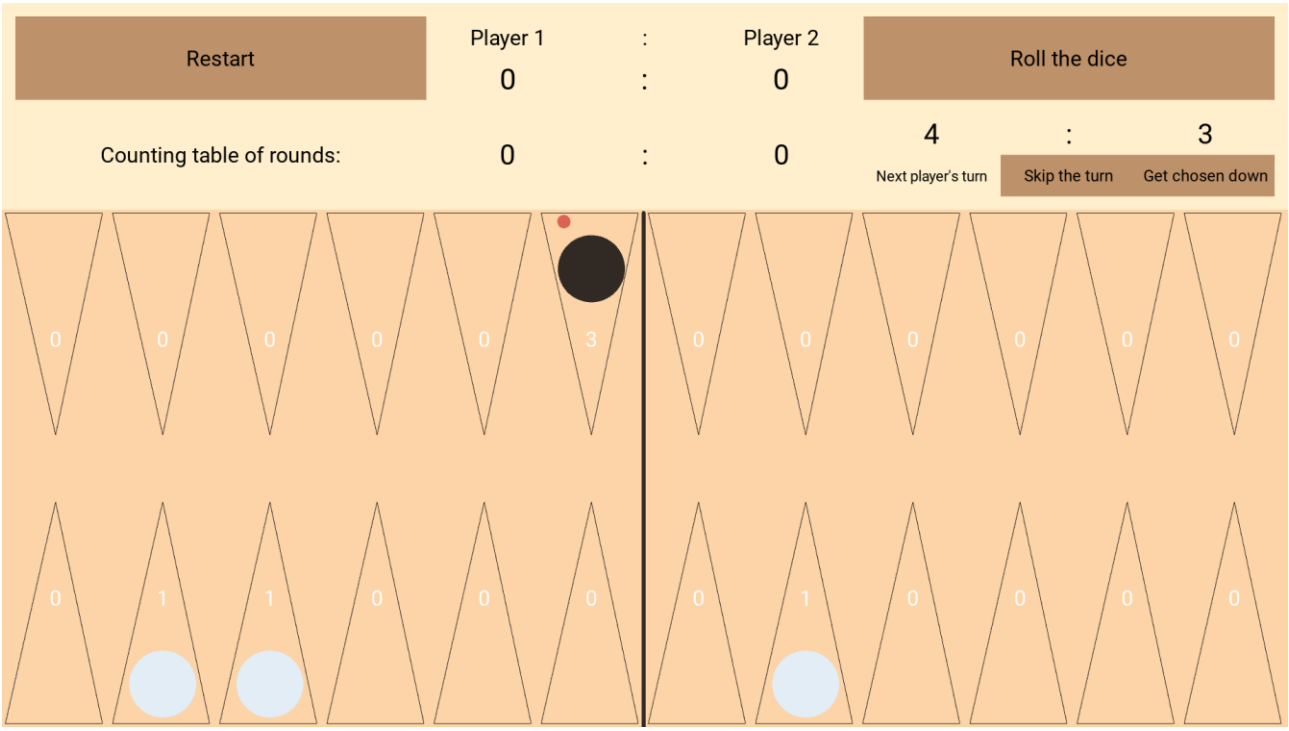

Рис. 4.7 Гра продовжується

З попереднього етапу ми переходимо до вже вирішеної ситуації, коли хід перейшов до наступного гравця. За схожим принципом відбувається скасування вибору конкретної фішки, наприклад коли гравець не може походити тією фішкою, яку він обрав спочатку або якщо він просто захотів змінити свій вибір.

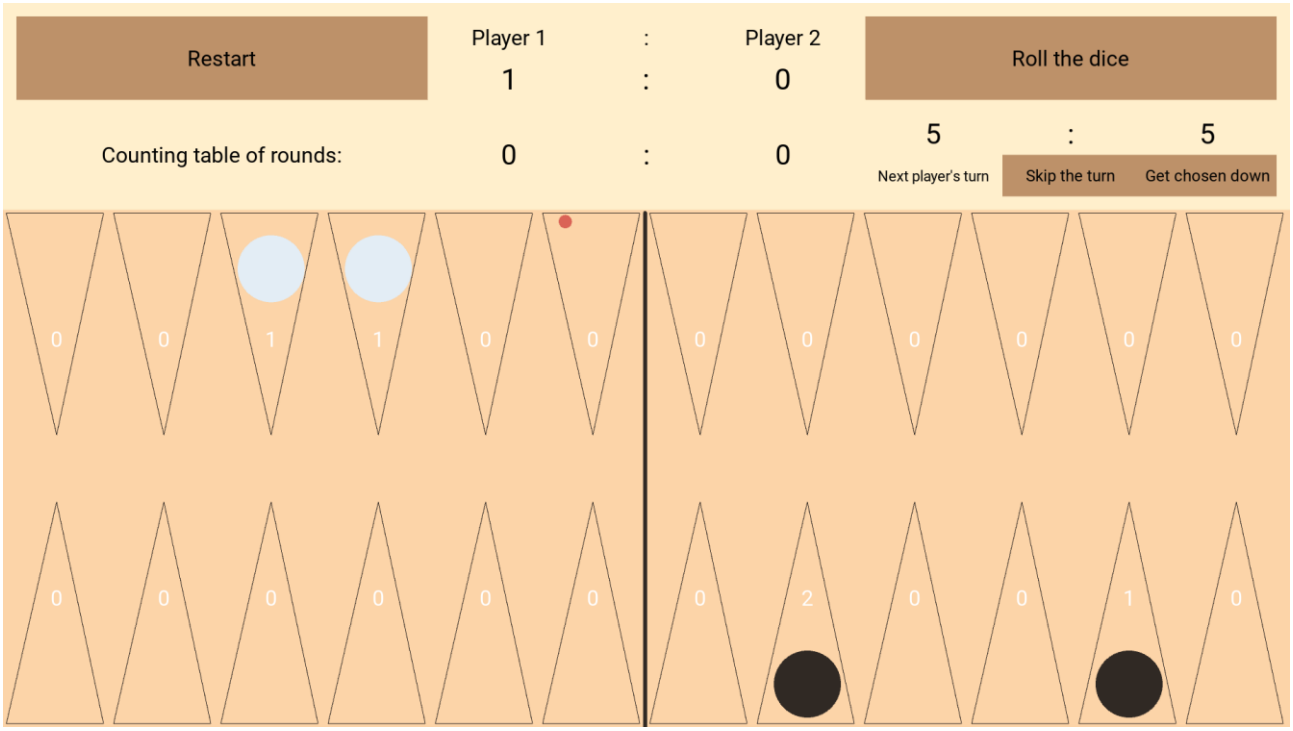

Рис. 4.8 Фінальний етап

Разом із досягненням фінального квартилю однією зі сторін з’являється можливість сховати фішки у «початковий стек», так як перемога залежить від того – хто найшвидше прибере свої фішки з поля. Також в процесі зникнення фішок буде збільшуватися рахунок таблиці Player 1 : Player 2, в якому відображається кількість схованих фішок кожним гравцем.

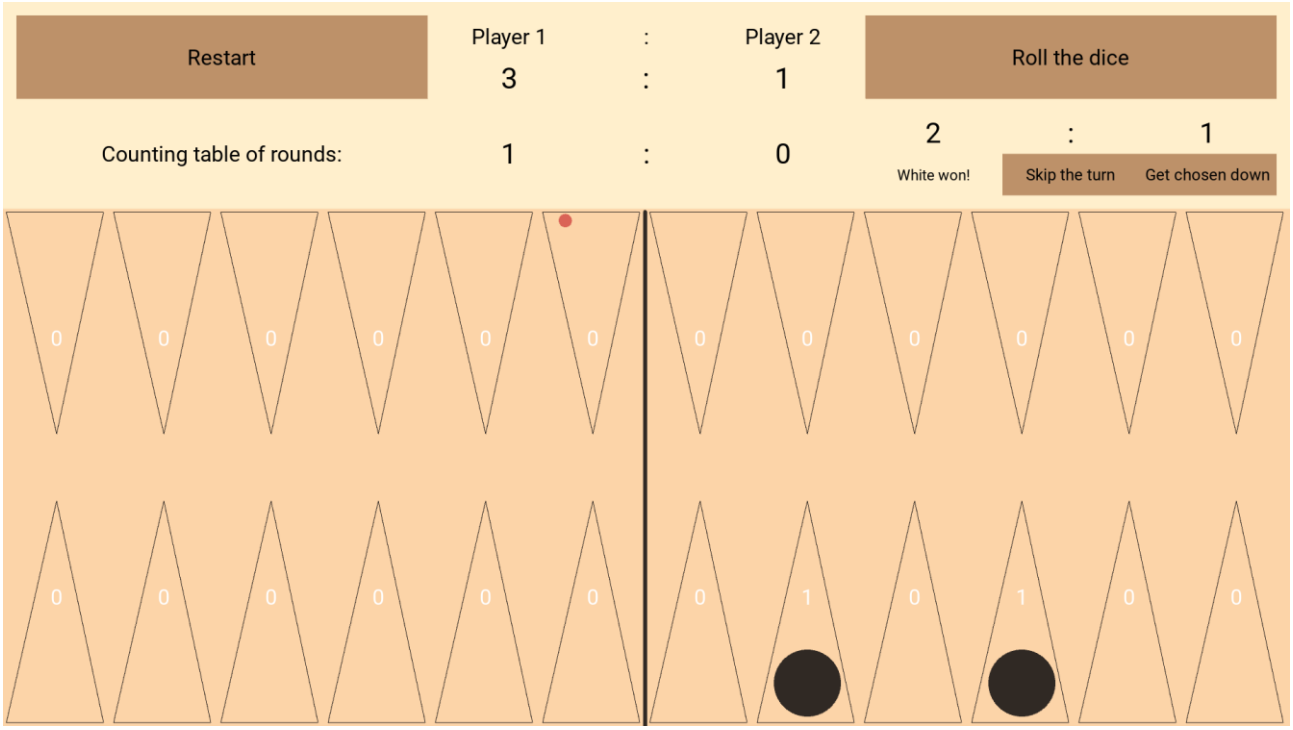

Рис. 4.9 Перемога білих

В даному тесті перемогу отримав гравець, що грав білими фішками, про що свідчить надпис «White won!», а також змінений рахунок таблиці «Counting table of rounds».

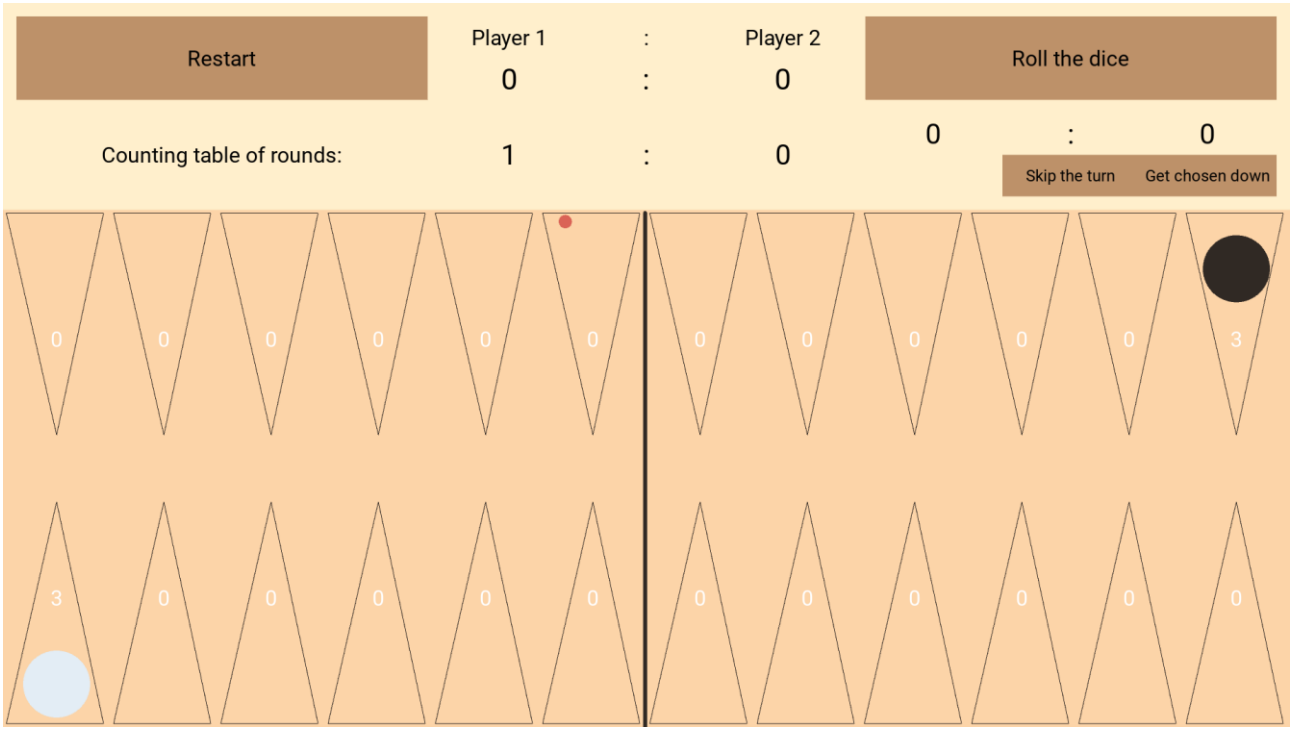

Рис. 4.10 Натиснення кнопки «Restart»

Як можемо побачити, після натиснення кнопки перезапуску гри фішки повернулися до початкових позицій, але рахунок таблиці залишився зміненим з урахування перемоги «білих».

Висновок

Виконуючи курсову роботу, я розробив інтерактивну гру «Нарди» з можливістю грати на ПК або ноутбуку. Під час виконання завдання я ознайомився з патерном програмування MVC, а також закріпив на практиці способи його використання у проектах та всі інші навички, отримані з курсу «Інженерія програмного забезпечення». Розроблена мною гра відповідає всім вимогам технічного завдання, відповідно до умов та правил справжньої гри «Нарди», а тому й успішно пройшла всі тестування та перевірки на відповідність вимогам.

Список використаних джерел

1. Python 3.12.3 documentation: https://docs.python.org/3/
2. Kivy 2.3.0 documentation: https://kivy.org/doc/stable/
3. Сайт зі зручним каталогом патернів програмування: https://docs.python.org/3/
4. Патерн «Спостерігач»: https://refactoring.guru/uk/design-patterns/observer
5. Методичні вказівки до виконання лабораторних робіт з дисципліни «Інженерія програмного забезпечення». Лабораторна робота No 4 с.22. А.І Антонюк, А.О. Болдак. Київ, 2022.
6. MVC сайт вікіпедії: https://pl.wikipedia.org/wiki/Model-View-Controller
7. Youtube посилання на курс, що дуже мені допоміг розібратися з модулем Kivy: https://youtube.com/playlist?list=PLCC34OHNcOtpz7PJQ7Tv7hqFBP_xDDjqg&si=jHcpDw2aNqhD65r8

Додаток А

Посилання на проект в Github:

https://github.com/VarRoman/Course_work_Nardy

Додаток В

Файл nardy.py:

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.anchorlayout import AnchorLayout
from kivy.uix.label import Label
from kivy.uix.widget import Widget
from kivy.lang import Builder
from kivy.graphics import Color, Rectangle, Ellipse, Line
from kivy.core.window import Window
from kivy.config import Config
from kivy.uix.gridlayout import GridLayout
from random import randint
from screeninfo import get_monitors

Builder.load_file('nardy.kv')

Config.set('graphics', 'resizable', '0')

total_score_white = 0
total_score_black = 0


class MyApp(App):  # Клас, що реалізує головний лейаут за допомогую метода build
    def build(self):
        Window.fullscreen = 'auto'
        self.layout = MainLayout()

        return self.layout


class MainLayout(BoxLayout):  # Головний лейаут, до якого додається контрольний пункт та ігрове поле
    def giving_the_dices(self, obj, value):  # Для отримання випадкових чисел з кубиків
        if value != 'down':
            self.gm.message_label = self.pl.children[0].children[0].children[0].children[2]
            self.gm.first_label_dice = self.pl.children[0].children[0].children[0].children[3]
            self.gm.second_label_dice = self.pl.children[0].children[0].children[0].children[5]
            self.gm.turning_dices = True

    def turning_down_the_checker(self, obj, value):  # Метод щоб переобрати вже обрану фішку
        if value != 'down':
            self.gm.chosen = None

    def skip_the_turn(self, obj, value):  # Метод для пропуску ходу, коли немає можливості ходити
        if value != 'down':
            if self.gm.player_turn == (.89, .93, .96, 1):
                self.gm.player_turn = (.19, .16, .14, 1)
            else:
                self.gm.player_turn = (.89, .93, .96, 1)
            self.gm.table_new = []
            self.gm.turning_dices = False

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
```

```
        # self.line_widget = Widget()
        self.orientation = 'vertical'
        self.size_hint = (1, 1)
        self.pl = PlayerPlace()
        self.gm = GamePlace()
        # self.rolling_button = self.pl.children[0].children[0].children[3]
        self.rolling_button = self.pl.children[0].children[0].children[3]
        self.rolling_button.fbind('state', self.giving_the_dices)

        self.rolling_button.state = 'down'

        self.giving_up_the_checker_button =
self.pl.children[0].children[0].children[0].children[0]
        self.giving_up_the_checker_button.fbind('state',
self.turning_down_the_checker)

        self.giving_up_the_turn_button =
self.pl.children[0].children[0].children[0].children[1]
        self.giving_up_the_turn_button.fbind('state', self.skip_the_turn)

        self.gm.white_player_final_label =
self.pl.children[0].children[0].children[4].children[2]
        self.gm.black_player_final_label =
self.pl.children[0].children[0].children[4].children[0]

        self.pl.children[0].children[0].children[1].children[2].text =
str(total_score_white)
        self.pl.children[0].children[0].children[1].children[0].text =
str(total_score_black)

        self.gm.white_player_final_record_label =
self.pl.children[0].children[0].children[1].children[2]
        self.gm.black_player_final_record_label =
self.pl.children[0].children[0].children[1].children[0]

        # self.gm.total_value_white = total_score_white
        # self.gm.total_value_black = total_score_black


        self.add_widget(self.pl)
        self.add_widget(self.gm)
        self.rolling_button.state = 'normal'

        self.monitor = []

        for m in get_monitors():
            self.monitor.append(m.width)
            self.monitor.append(m.height)

        with self.canvas:
            Color(.19, .16, .14, 1)  # Колір лінії (червоний)
            Line(points=[self.monitor[0] / 2 - 2, 0, self.monitor[0] / 2 - 2,
self.monitor[1] * .71], width=3)


class PlayerPlace(BoxLayout):  # Верхня частина (контрольний пункт)
    def restart_the_game(self):  # метод для перезапуску гри
        Window.clear()
        App.get_running_app().stop()
        MyApp().run()

    def get_random_points(self):
        return str(randint(1, 6))
```

```
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.total_score_white = str(total_score_white)
        self.total_score_black = str(total_score_black)


class Checker(Widget):  # Віджет фішки
    def update_checker_canvas(self, new_color):
        self.color = new_color
        self.canvas.clear()
        with self.canvas:
            Color(*self.color)
            Ellipse(size=[100, 100], pos=(self.x - (self.height / 2), self.y))

    def __init__(self, color, position, counter, x, y, **kwargs):
        super().__init__(**kwargs)
        self.x = x
        self.y = y
        self.counter = counter
        self.color = color
        self.position = position
        self.label = Label(text=f'{self.counter}', font_size='32')

        with self.canvas:
            Color(*self.color)
            Ellipse(size=[100, 100], pos=(self.x - (self.height / 2), self.y))


class GamePart(AnchorLayout):  # Частини Gameplace, в яких буде розміщено
Checkers, а також трикутники для візуалу
    def on_size(self, *args):  # метод, який не дає трикутник розпадатися
        self.canvas.clear()
        with self.canvas:
            Color(.99, .83, .66, 1)
            Rectangle(size=self.size, pos=self.pos)
            Color(49 / 255, 41 / 255, 36 / 255, 1)

            if self.place:
                Line(points=(self.pos[0] + 5, self.height + self.pos[1] - 5,
self.width + self.pos[0] - 10,
                             self.height + self.pos[1] - 5, self.pos[0] +
(self.width / 2), self.pos[1] + 50),
                     close=True, size=self.size)

            else:
                Line(points=(
                    self.pos[0] + 5, self.pos[1] + 5, self.width + self.pos[0] -
10, self.pos[1] + 5, self.pos[0] +
                    (self.width / 2), self.height - 50), close=True,
size=self.size)

        if self.place and self.place != -1:
            self.ch = Checker(self.color, self.position, self.counter,
                              self.x + self.width / 2, self.height * 1.5 + 55)
            self.add_widget(self.ch.label)
            self.add_widget(self.ch)
        else:
            self.ch = Checker(self.color, self.position, self.counter,
                              self.x + self.width / 2, 14)

            self.add_widget(self.ch.label)
            self.add_widget(self.ch)

    def __init__(self, place, position, color, counter, **kwargs):
```

```
        super().__init__(**kwargs)
        self.place = place  # змінна для визначення місця canvas

        self.position = position  # який номер цієї позиції відносно дошки
        self.color = color  # який колір фішок, що знаходяться на цьому полі
        self.counter = counter  # кількість фішок на цьому полі


class GamePlace(GridLayout):  # Ігрове поле, на якому будуть відбуватися
переміщення фішок
    def on_touch_down(self, touch):  # Один великий метод для налаштування
механіки гри та її правил,
        # який спрацьовує під час натискання
        global total_score_white, total_score_black
        if not self.blocked:
            if not sum([j.counter for j in self.place_on_game if j.color ==
(.89, .93, .96, 1)]):  # Перевірка чи вже не кінець
                self.message_label.text = "White won!"
                total_score_white += 1
                self.white_player_final_record_label.text =
f"{total_score_white}"
                self.blocked = True

            if not sum([j.counter for j in self.place_on_game if j.color ==
(.19, .16, .14, 1)]):
                self.message_label.text = "Black won!"
                total_score_black += 1
                self.black_player_final_record_label.text =
f"{total_score_black}"
                self.blocked = True

            if not self.chosen:  # Перевірка для визначення вибору фішки гравця
                for i in self.place_on_game:
                    if i.collide_point(*touch.pos) and i.counter and i.color ==
self.player_turn:
                        self.chosen = i
                        break

            else:
                for i in self.place_on_game:  # Перевірка на знаходження
необхідного місця до переміщення фішки
                    if (i.collide_point(*touch.pos) and (i.color ==
self.chosen.color
                        or i.color == (.99, .83, .66, 1)) and i != self.chosen
and self.turning_dices):
                        if not self.table_new:  # Перевірка наявності ходів
                            self.message_label.text = ""

self.table_new.extend([int(self.first_label_dice.text),
int(self.second_label_dice.text)])
                            if self.table_new[0] == self.table_new[1]:
                                self.table_new.extend([self.table_new[0],
self.table_new[1]])

                        if (i.position - self.chosen.position in self.table_new
or  # Перевірка правильності ходу
                                (i.position + 24) - self.chosen.position in
self.table_new):
                            # Перевірка на фінальний етап для данного гравця у
грі
                            if ((self.chosen.color == (.89, .93, .96, 1) and
self.chosen.position in range(6, 12) and
                                sum([j.counter for j in
self.place_on_game[6:12]]) != self.counter_white) and
```

```
                                    i.position not in range(6, 12)):
                                self.chosen = None
                                break

                        # Те саме, але вже для гравця чорними
                        elif ((self.chosen.color == (.19, .16, .14, 1) and
self.chosen.position in range(18, 24) and
                                    sum([j.counter for j in
self.place_on_game[18:]]) != self.counter_black) and
                                    i.position not in range(18, 24)):
                                self.chosen = None
                                break

                        # Перевірка на фінальний етап з підтвердженням для
чорних
                        elif ((self.chosen.color == (.19, .16, .14, 1) and
self.chosen.position in range(18, 24) and
                                    sum([j.counter for j in
self.place_on_game[18:]]) == self.counter_black) and
                                    i.position not in range(18, 24)):

                                self.black_player_final_label.text =
f"{int(self.black_player_final_label.text) + 1}"
                                self.counter_black -= 1

                        # Перевірка на фінальний етап з підтвердженням для
білих
                        elif ((self.chosen.color == (.89, .93, .96, 1) and
self.chosen.position in range(6, 12) and
                                    sum([j.counter for j in
self.place_on_game[6:12]]) == self.counter_white) and
                                    i.position not in range(6, 12)):

                                self.white_player_final_label.text =
f"{int(self.white_player_final_label.text) + 1}"
                                self.counter_white -= 1

                        # Зміни при звичайному ході
                        else:
                            i.counter = i.counter + 1
                            i.ch.counter = i.counter
                            i.ch.label.text = f'{i.counter}'

                            if i.color == (.99, .83, .66, 1) and i.counter >
0:
                                i.color = self.chosen.color
                                i.on_size()

                        # Загальна процедура для проходження фішки та зміни
всіх необхідних елементів
                        self.chosen.counter = self.chosen.counter - 1
                        self.chosen.ch.counter = self.chosen.counter
                        self.chosen.ch.label.text = f'{self.chosen.counter}'

                        if self.chosen.counter == 0:
                            self.chosen.color = (.99, .83, .66, 1)
                            self.chosen.on_size()
                        if (i.position - self.chosen.position) in
self.table_new:
                            self.table_new.remove(i.position -
self.chosen.position)
                        else:
                            self.table_new.remove(24 + i.position -
self.chosen.position)
```

```
                            if not self.table_new:
                                self.message_label.text = "Next player's turn"
                                if self.player_turn == (.89, .93, .96, 1):
                                    self.player_turn = (.19, .16, .14, 1)
                                else:
                                    self.player_turn = (.89, .93, .96, 1)
                                self.turning_dices = False

                            self.chosen = None
                            break
                    if not self.turning_dices:
                        self.message_label.text = "Next player should\n turn the
dices!"

    def __init__(self, **kwargs):  # Налаштування цілої купи параметрів, велика
частина з яких будуть задаватися ззовні
        super().__init__(**kwargs)
        self.turning_dices = False
        self.player_turn = (.89, .93, .96, 1)
        self.table_new = []
        self.counter_white = 15
        self.counter_black = 15

        self.first_label_dice = None
        self.second_label_dice = None
        self.message_label = None

        self.white_player_final_label = None
        self.black_player_final_label = None

        self.white_player_final_record_label = None
        self.black_player_final_record_label = None

        self.blocked = False

        self.rows = 2
        self.cols = 12

        self.place_on_game = []
        self.chosen = None

        # Встановлення базових фішок фонового кольору та 0 тексту Label для
подальших маніпуляцій з ними
        for i in range(12):
            self.place_on_game.append(GamePart(1, 11 - i, (.99, .83, .66, 1),
0))
            self.add_widget(self.place_on_game[i])

        self.place_on_game.reverse()

        for i in range(12, 24):
            self.place_on_game.append(GamePart(0, i, (.99, .83, .66, 1), 0))
            self.add_widget(self.place_on_game[i])

        self.place_on_game[0].counter = 15
        self.place_on_game[0].color = (.19, .16, .14, 1)
        self.place_on_game[12].counter = 15
        self.place_on_game[12].color = (.89, .93, .96, 1)


if __name__ == '__main__':
    MyApp().run()
```

Файл nardy.kv:

```
<PlayerPlace>
    size_hint: (1, .4)
    canvas:
        Color:
            rgba: (255/255, 239/255, 204/255, 1)
        Rectangle:
            size: self.size
            pos: self.pos
    BoxLayout:
        orientation: 'vertical'
        size: self.size
        padding: 20
        spacing: 20

        GridLayout:
            cols: 3
            rows: 2
            spacing: 20

            Button:
                text: "Restart"
                size_hint: (.5, .3)
                background_color: (.74, .57, .41, 1)
                background_normal: ''
                font_size: 32
                color: 'black'
                on_press: root.restart_the_game()

            GridLayout:
                size_hint: (.5, .3)
                cols: 3
                rows: 2
                font_size: 32

                Label:
                    text: 'Player 1'
                    font_size: 32
                    color: 'black'

                Label:
                    text: '  :  '
                    font_size: 32
                    color: 'black'

                Label:
                    text: 'Player 2'
                    font_size: 32
                    color: 'black'

                Label:
                    text: '0'
                    font_size: 44
                    color: 'black'

                Label:
                    text: '  :  '
                    font_size: 44
                    color: 'black'

                Label:
                    text: '0'
                    font_size: 44
```

```
                color: 'black'

        Button:
            text: "Roll the dice"
            size_hint: (.5, .3)
            background_color: (.74, .57, .41, 1)
            background_normal: ''
            font_size: 32
            color: 'black'
            on_press:
                first_cube.text = root.get_random_points()
                second_cube.text = root.get_random_points()

        Label:
            size_hint: (.2, .2)
            font_size: 32
            color: 'black'
            text: 'Counting table of rounds:'

        GridLayout:
            size_hint: (.5, .3)
            cols: 3
            rows: 2
            font_size: 32

            Label:
                text: '0'
                font_size: 42
                color: 'black'

            Label:
                text: ':'
                font_size: 42
                color: 'black'

            Label:
                text: '0'
                font_size: 42
                color: 'black'

        GridLayout:
            size_hint: (.5, .3)
            cols: 3
            rows: 2
            font_size: 32

            Label:
                id: first_cube
                text: '0'
                font_size: 42
                color: 'black'

            Label:
                text: ':'
                font_size: 42
                color: 'black'

            Label:
                id: second_cube
                text: '0'
                font_size: 42
                color: 'black'

            Label:
```

```
                    text: ''
                    font_size: 22
                    color: 'black'

                Button:
                    text: "Skip the turn"
                    background_color: (.74, .57, .41, 1)
                    background_normal: ''
                    font_size: 24
                    color: 'black'

                Button:
                    text: "Get chosen down"
                    background_color: (.74, .57, .41, 1)
                    background_normal: ''
                    font_size: 24
                    color: 'black'
```