ЗМІСТ

[ВСТУП 6](#_Toc437509982)

[1 АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАВДАННЯ НА РОБОТУ 8](#_Toc437509983)

[1.1 Предметна область 8](#_Toc437509984)

[1.2 Існуючі реалізації 9](#_Toc437509985)

[1.3 Розробка технічного завдання на роботу 11](#_Toc437509986)

[1.4 Обґрунтування вибору мови програмування 13](#_Toc437509987)

[1.5 Висновки 14](#_Toc437509988)

[2 РОЗРОБКА ІНТЕРФЕЙСУ ПРОГРАМИ ТА ПРИКЛАДІВ ВИКОРИСТАННЯ 15](#_Toc437509989)

[2.1 Розробка інтерфейсу програми 15](#_Toc437509990)

[2.2 Приклади використання 16](#_Toc437509991)

[3 РОЗРОБКА ДІАГРАМ КЛАСІВ, ОБ’ЄКТІВ ТА СТАНУ 19](#_Toc437509992)

[3.1 Діаграми класів 19](#_Toc437509993)

[3.1.1 Діаграми наслідування 19](#_Toc437509994)

[3.1.2 Діаграми агрегації 20](#_Toc437509995)

[3.1.3 Діаграми композиції 20](#_Toc437509996)

[3.1.4 Діаграми асоціації 21](#_Toc437509997)

[3.2 Діаграми кооперації 22](#_Toc437509998)

[3.3. Діаграми послідовності 23](#_Toc437509999)

[3.4 Діаграми стану 24](#_Toc437510000)

[4 РОЗРОБКА ПІДСИСТЕМИ ГРАФІЧНОГО ВІДОБРАЖЕННЯ 25](#_Toc437510001)

[4.1 Модель графічного відображення 25](#_Toc437510002)

[4.2 Графічні процедури підсистеми графічного відображення 26](#_Toc437510003)

[5 ВИКОРИСТАННЯ ЗАСОБІВ ПРОГРАМУВАННЯ JDK 27](#_Toc437510004)

[5.1 Функції JDK для роботи з таймером. 27](#_Toc437510005)

[6 РОЗРОБКА ПІДСИСТЕМИ СЕРІАЛІЗАЦІЇ/ДЕСЕРІАЛІЗАЦІЇ ДАНИХ 29](#_Toc437510006)

[6.1 Використання інтерфейсу Serializable 29](#_Toc437510007)

[6.2 Механізми введення виведення мови Java 30](#_Toc437510008)

[7 КЕРІВНИЦТВО КОРИСТУВАЧА 31](#_Toc437510009)

[7.1 Встановлення програми та необхідні файли/каталоги 31](#_Toc437510010)

[7.2 Запуск програми 31](#_Toc437510011)

[7.3 Користувацький інтерфейс 31](#_Toc437510012)

[ВИСНОВКИ 34](#_Toc437510013)

[ПЕРЕЛІК ПОСИЛАНЬ 35](#_Toc437510014)

[Д О Д А Т К И 36](#_Toc437510015)

[Додаток А](#_Toc437510016). [Лістинг модуля Organization.java 37](#_Toc437510017)

[Додаток Б](#_Toc437510018). [Лістинг модуля MagicOrganization.java 39](#_Toc437510019)

[Додаток B](#_Toc437510020). [Лістинг модуля OrdenOfThePhoenix.java 45](#_Toc437510021)

[Додаток Г](#_Toc437510022). [Лістинг модуля TheDeathEaters.java 47](#_Toc437510023)

[Додаток Д](#_Toc437510024). [Лістинг модуля MagicEntity.java 48](#_Toc437510025)

[Додаток Е](#_Toc437510026). [Лістинг модуля HeaderMagEntity.java 51](#_Toc437510027)

[Додаток Ж](#_Toc437510028). [Лістинг модуля StudentMagEntity.java 52](#_Toc437510029)

[Додаток З](#_Toc437510030). [Лістинг модуля AnimalMagicEntity.java 53](#_Toc437510031)

[Додаток И](#_Toc437510032). [Лістинг модуля EntityJPanel.java 54](#_Toc437510033)

[Додаток Й](#_Toc437510034). [Лістинг модуля MainWindow.java 57](#_Toc437510035)

[Додаток К](#_Toc437510036). [Лістинг модуля EntityCreator.java 60](#_Toc437510037)

# ВСТУП

Магія (англ. Magic, а також: «чари», «чаклунство», «ведунство», «чарування» і застаріле «волхвування») - здатність, завдяки якій за допомогою внутрішньої (в основному психічної) енергії можна управляти предметами і явищами. Магія є надприродною силою, яка може змінювати тканину реальності на фундаментальному рівні.

Магічна істота – будь-яке живе створіння, яке має можливість використовувати магію. Це є спадковою рисою і передається своїм нащадкам від пращурів. У світі магів існують багато чарівних істот. Умовно їх можна розділити на тих, хто належить до світу фауни і тих, хто так чи інакше наділений розумом.

У людей вміння чи невміння творити магію визначається генетичною спадковістю. У пари чарівників є нормою наявність у їхніх дітей чарівництва, але у подружньої пари маглів це явище досить рідкісне. Пов'язано це з тим, що ген магії, присутній в ДНК відьом і чаклунів, є домінантним. Сквібами називають людей, народжених в чарівних сім'ях, але не здатних творити магію. Це трапляється, коли у нащадка чарівників такий ген з якоїсь причини не проявляється, в результаті чого він стає нездатним до чарівництва.

Відьма або чарівник, народжені маглівськими батьками, відомі як маглонароджені. Це трапляється, коли сім'я маглів походить від сквіба, у нащадків якого ген магії раптом стає активним багато поколінь по тому. Маглонароджених чарівників набагато більше, ніж сквібів, але це може бути пов'язано з непорівнянними масштабами популяцій світу звичайних людей і світі чарівництва.

Більшості магів для концентрації магічної енергії в певній точці необхідна чарівна паличка, жезл або чарівний посох. Дуже небагато обходяться всього лише пасами рук або навіть просто поглядом, не кажучи вже про віртуозів, які здатні творити чудеса, навіть не відволікаючись від буденних справ. У світі «Гаррі Поттера» магія - невід'ємна частина всієї Поттеріани.

Для безпосередньої реалізації своїх насущних магічних бажань маги використовують заклинання. Більш складні магічні дії виконуються за допомогою магічних ритуалів. Один з таких ритуалів використовував Лорд Волдеморт для свого відродження.

Темні мистецтва - це заклинання і магічні практики, задумані як здатні заподіяти відчутної шкоди іншим людям. Використовуються, як правило, зі злими намірами. Чарівників, що використовують Темні мистецтва, називають темними магами, наймогутніший з яких Лорд Волдеморт.

Чарівники, в більшості своїй вважають себе вершиною еволюції, з упередженням ставляться до інших чарівних рас і в гордині своїй відносять їх до розумних істот, а то і до тварин, наділених лише проблисками розуму. Тим часом, магія розумних чарівних істот у багатьох випадках недоступна чарівникам. До розумних чарівних істот маги відносять вампірів, віїл, велетнів, гоблінів, домашніх ельфів, кентаврів, перевертнів, русалок, тритонів і сфінксів. Також до них можна віднести, ймовірно, тролів і акромантулів.

Поціновувачам фентезійного жанру в літературі буде цікаво дізнатись більше про Поттеріану і прийняти участь в магічних дуелях. Саме тому для теми курсової роботи було обрану систему симуляції протидії магічних істот.

# 1 АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАВДАННЯ НА РОБОТУ

## 1.1 Предметна область

Магічні істоти, які так чи інакше наділені розумом, мають власні потреби та бажання впродовж свого життя. Для їх задоволення вони можуть користуватись різними способами, одні з яких відповідають законам моралі, інші не дуже, а деякі взагалі протирічать (до таких відносяться Темні мистецтва).

Відьми та чаклуни, як представники соціальних магічних істот, можуть об’єднуватись в певні групи чи організації для досягнення загальної мети. Інші магічні істоти не схильні до створення таких складних структур, як організації, проте вони можуть ставати на сторону тих чи інших людських організацій у відповідності до своїх потреб.

У зв’язку з діяльністю могутнього темного мага Лорда Волдеморта, в світі «Гаррі Поттера» утворилось дві магічні організації: Смертежери та Орден Фенікса.

Смертежери (англ. Death Eaters, існує також переклад «пожирачі смерті») - група темних чарівників, послідовників Лорда Волдеморта, що борються в якості елітних бійців в першій і другій магічних війнах.

Символ Смертежерів - Темна мітка, череп з виповзає з рота змією. Кожен з них має на лівій руці точно таку ж магічну мітку, що служить засобом зв'язку з Волан-де-Мортом. У роки Першої Війни Темна мітка запускалася в небо над будинками убитих Пожирачами Смерті за допомогою заклинання Морсмордре.

Смертежери влаштовували безладдя, нападали без причин на маглів, вбивали їх, причому нерідко це були масові вбивства. Магів, що відмовилися перейти на бік Волан-де-Морта, часто теж вбивали або ж піддавали закляттям Імперіус. Сім'ї убитих магів часто знищувалися.

Орден Фенікса (англ. Order of the Phoenix) - організація, заснована Альбусом Дамблдором для боротьби з лордом Волан-де-Мортом і Пожирачами Смерті.

Вперше Орден згадується в п'ятій книзі. Мета Ордена - захист магів і маглів від смертежерів і боротьба з Волан-де-Мортом. Іноді вони дублюють (набагато більш успішно) роботу мракоборцем Міністерства магії.

Орден зазвичай діє таємно, навіть від міністерства, оскільки офіційна позиція нерідко йде врозріз з цілями Ордену. Багато членів Ордена не афішують свою зв'язок з феніксовцями.

## 1.2 Існуючі реалізації

На сьогоднішній день існує декілька комп’ютерних ігор по світу Гаррі Поттера. Найвідоміші з них це:

Серія ігор «Гаррі Поттер». В кожній частині якої необхідно проходити героями по сюжету, відповідно до сюжету книги. На їхньому шляху будуть траплятись інші магічні істоти, від яких необхідно захистити головних героїв, використовуючи магію та їх уміння (рис. 1.1).



Рисунок 1.1 – Фрагмент гри «Гаррі Поттер та в’язень Азкабану»

Гра «Wonderbook: Книга заклять», в якій необхідно вивчати та відтворювати закляття за допомогою спеціального контролера, який грає роль чарівної палички (рис. 1.2).



Рисунок 1.2 – Фрагмент гри «Wonderbook: Книга заклять»

Гра «Гаррі Поттер для Kinnect», в якій необхідно пройти всі роки навчання Гаррі Поттера в Хогвартсі (рис. 1.3).



Рисунок 1.3 – Фрагмент гри «Гаррі Поттер для Kinnect»

Недоліком даних реалізацій є те, що в них, по-перше, занадто складна логіка та масштаби ігрового світу, по-друге, немає можливості відтворити військові дії між магічними організаціями та керувати діями будь-якого з їх членів, по-третє, деякі з цих реалізацій не доступні для персональних комп’ютерів. Тому розробка власної програмної реалізації, в якій є можливість керувати персонажами, створювати власних, переміщувати їх між організаціями є актуальною задачею.

## 1.3 Розробка технічного завдання на роботу

Потрібно створити Java-додаток, який реалізує: класи мікро та двох макрооб’єктів різних класів згідно варіанту курсової.

Клас мікрооб’єкту повинен містити не менше 4 елементів змінних і не менше 4 методів. Як мінімум одна змінна повинна бути типу int, одна – типу double і як мінімум одна – рядок (будь якої реалізації). В класі повинен бути присутній конструктор, а також як мінімум один елемент, що створений динамічно. Це може бути массив, структура, інший об’єкт або навіть одна змінна. Повинен бути присутній код, що створює цей елемент – наприклад у конструкторі.

Повинно бути реалізоване глибинне копіювання шляхом перевантаження копіюючого конструктора, три випадки виклику копіюючого конструктора.

Повинно також бути продемонстровано створення об’єкту статично та динамічно; додано у клас одну статичну змінну та одну статичну функцію – та продемонстровано їх використання. Повинно також бути продемонстровано використання наступних механізмів мови Java: ініціалізатор конструктора, оператор this, типи доступу public, private, protected.

Зображення мікрооб’єкту повинно складатись не менше чим з трьох графічних примітивів, при цьому один з них повинен бути текст. Повинно також бути продемонстровано створення динамічного масиву мікрооб'єктів та відображення його у вікні програми. При натискуванні правої кнопки миші на мікрооб’єкті він повинен ставати активним/неактивним. При натискувань клавіш-стрілок активні об’єкти повинні рухатись у вікні програми.

Повина бути реалізована можливість додавати мікрооб’єкти шляхом натискування клавіши Insert. При цьому повинне з’являтись діалогове вікно, яке визначить параметри створюваного мікрооб’єкта. Крім керуючого елемента Button у діалоговому вікні також повинні бути використані як мінімум два з наступного списку: TextBox, CheckBox, ListBox, RadioButton. При натискувань клавіши Delete активні об’єкти повинні знищуватись. Якщо активного об’єкта нема – клавіша Delete ігнорується. Клавіша Esc повинна відміняти активацію обєкта.

Повинно бути реалізовано наслідування мікрооб’єктів – як мінімум три рівня наслідування. Зображення об'єктів класів нащадків повинно відрізнятись від зображень об’єктів базових класів. Функцію Draw слід зробити віртуальною. Крім функції Draw в класах мікрооб'єктв повинна бути присутня ще як мінімум одна віртуальна функція. Зображення кожного з макрооб’єктів може складатись з графічних примітивів: точка, лінія, прямокутник, коло, арка, полігон, текст, графічний образ. Зображення кожного макрооб’єкту повинно складатись не менше чим з трьох елементів, при цьому один з елементів повинен бути текст. Додати класи макрооб’єктів і команди створення самих макрооб’єктів і команди їх рисування в програму.

В програмі повинен бути як мінімум один макрооб’єкт кожного з двох класів макрооб'єктів. При цьому мікрооб'єкти повинні чітко розділитись на тих, які належать до макрооб’єктів і тих, які їм не належать (найбільш наглядно цен зобразити геометрично – ті що належать знаходяться всередині макрооб’єкта, ті що не належать - ззовні).

Рух мікрооб’єкта всередині макрооб’єкта – не обов’язковий. Повинна бути реалізована можливість рухати якимись чином макрооб’єкт (разом з всіма мікрооб'єктами, які йому належать). Повинна бути реалізована можливість включати виділені/виділений мікрооб’єкти/мікрооб’єкт в макроб’єкт та – навпаки – виключати виділені/виділений мікрооб’єкти/мікрооб’єкт з макроб’єкту. Можливість виділення декількох мікроб’єктів не є обов’язковою.

Мікроб’єкти повинні бути як мінімум у трьох станах: А) Належать макрооб’єкту першого типу; Б) Належать макрооб’єкту другого типу; В) Не належать жодному макрооб’єкту. Не забороняється мікроооб’єкту одночасно належати декільком макрооб’єктам одночасно. Повинен бути реалізований автоматичний рух деяких мікрооб’єктів зі зміною стану (див. А) Б) В) ). По натискуванню клавіши миші/клавіши на клавіатурі/команді меню характер руху повинен змінюватись.

Повинна бути запрограмована серіалізація/де-серіалізація всіх об’єктів у файл, яка обов’язково повинна зберігати не тільки власне інформацію про стан макро- та мікро-об’єктів, але й про їх позицію на екрані. При серіалізації/де-серіалізації обов’язково повинні використовуватись діалогові вікна, щоб запитати у користувача ім’я файлу (функції GetOpenFileName та GetSaveFileName).

У проекті повинні запрограмовані такі відносини між класами та об'єктами, які можуть бути наглядно проілюстровані на діаграмах агрегації, композиції, асоціації, використання, взаємодії, послідовності та стану.

Для нормального виконання програми необхідно наступне апаратне забезпечення:

1. Pentium 3 з частотою 2.0 GHz;
2. 256МБ оперативної пам’яті;
3. графічний адаптер SVGA (Super Video Graphic Adapter);
4. відеокарта об’ємом пам’яті не менше 64МБ;
5. клавіатура, ОС Windows ХР /Vista/7/8.
6. встановлена Java Virtual Machine

Розмір дискового простору, що займає програма: 16 704 байт.

Розмір оперативної пам’яті, що займає програма: 3 500 КБайт.

## 1.4 Обґрунтування вибору мови програмування

Для якісного і об`єктивного вибору мови програмування, потрібно порівняти найбільш актуальні мови програмування. Після цього визначити всі переваги і недоліки кожної з мов програмування. На сучасному розвитку найбільш актуальним і прогресивними мовами програмування є: С++, С# та Java.[1]

Таблиця 1.1 - Порівняння Java, С++, С#

| Особливість | Java | C + + | C # |
| --- | --- | --- | --- |
| Збір сміття | Так | Ні | Так |
| Кросплатформеність | Так | Так | Ні |
| Поліморфізм по замовчуванню | Так | Ні | Ні |
| Шаблони функцій | Так | Так | Ні |
| Foreach | Так | Ні | Так |
| Гарантована ініціалізація | Так | Ні | Так |
| Автоматичний деструктор | Ні | Так | Так |
| Множинне успадкування | Ні | Так | Ні |
| Перевантаження операторів | Ні | Так | Так |

В таблиці (таб. 1.1) наведено порівняння найбільш актуальних об’єктно-орієнтованих мов програмування на даному етапі розвитку інформаційних технологій.

Мова програмування Java виявилась найбільш підходящою для розробки швидкодіючої, кросплатформенної програми, так як має більше переваг, ніж інші мови, відібрані для порівняльного аналізу. Тому саме мову Java було обрано для розробки програми.

## 1.5 Висновки

Проаналізувавши існуючі аналоги продуктів по Поттеріані, було прийнято рішення створити власний програмний продукт, який буде реалізувати відсутні можливості інших програм.

Створено вимоги до програмного продукту, який буде реалізовано. Найефективнішою мовою реалізацію було обрано Java після проведення порівняльного аналізу серед інших найбільш популярних об’єктно-орієнтованих мов програмування.

# 2 РОЗРОБКА ІНТЕРФЕЙСУ ПРОГРАМИ ТА ПРИКЛАДІВ ВИКОРИСТАННЯ

## 2.1 Розробка інтерфейсу програми

На головному вікні програми відображається внутрішній світ, що симулюється у програмі, його мікрооб’єкти [2] та макрооб’єкти (рис. 2.1). Після цього програма очікує дій від користувача. Паралельно, у окремому потоці відбувається обробка сигналів з таймеру.

Орден фенікса

Смертежери

Робоча область

Симуляція протидії магічних істот

Акт.:Смертежер, маг Стрілки/Ins/Del/Esc/1/2/3/F2

Рисунок 2.1 – Структура вікна програми

Назва вікна - Симуляція протидії магічних істот

Усі об’єкти програми відображуються на робочій області та можуть рухатись в її межах. Панелі складаються з двох прямокутників, один з яких більший за інший та виконує роль рамки. На них відображається текст – назва панелі. В центрі зображена фігура, овал в панелі «Орден Фенікса» та прямокутник з скругленими кутами в панелі «Смертежери».

Для додавання нових істот потрібно активувати одну з панелей, після чого натиснути клавішу Insert, що відкриє вікно додавання істоти, або використати гарячі клавіші: 1 – маг, 2 – студент школи чародійства, 3 – магічне створіння.

Рух елементів програми можна здійснювати за допомогою клавіш Left - ліворуч, Right - праворуч, Up - вверх, Down – вниз. Для цього потрібно активувати елемент, тобто натиснути на нього лівою кнопкою миші, після чого колір рамки елемента зміниться. У істот, які належать до Ордена Фенікса рамка стане синього кольору, у Смертежерів – червоного, у нейтральних істот – жовтого. Колір рамки панелей макрооб’єктів стане фіолетового кольору. Активний макрооб’єкт рухається разом з мікрооб’єктами, які йому належать. Одночасно можна активувати декілька елементів.

Щоб видалити елементи, необхідно виділити їх, активувати панель, до якої вони належать та натиснути клавішу Delete. Клавіша Escape деактивує всі елементи активної панелі. Клік правою кнопкою миші викликає спливаюче меню, яке дозволяє змінити належність до організації виділених об’єктів активної панелі [3].

Меню збереження та відкриття файлів викликається клавішою F2, у ньому обирається шлях до файлу, у який потрібно зберегти кожен з макрооб’єктів чи завантажити його.

Після вибору розташування файлів (не обов’язково для усіх макрооб’єктів) потрібно натиснути кнопку «Зберегти» або «Завантажити».

## 

## 2.2 Приклади використання

Приклади використання (use cases) призначені для показу можливостей програми користувачам, сприяє зручному використанню продукту та поліпшенню його якості.

Приклад 1. Створення нової магічної істоти + діалогове вікно (рис. 2.2)

Після натиснення кнопки Insert з’являється діалогове вікно [4], у якому користувач має обрати з поданих списків належність до організації та магічний рівень істоти. Також є можливість обрати заготовлене ім’я істоти чи ввести вигадане. Після натиснення клавіші «Створити» істота з’явиться у випадковому місці на панелі магічної організації, до якої він належить.

Барті Крауч

Маг

Нейтральний

Сила

Ім’я

Рівень

Організація

Робоча область

Симуляція протидії магічних істот

Акт.:Смертежер, маг Стрілки/Ins/Del/Esc/1/2/3/F2

Рисунок 2.2 – Створення нового магічної істоти і діалогове вікно

Приклад 2. Взаємодія магічної істоти з організаціями.

Магічна істота може змінювати свою належність до магічної організації.

Робоча область

Магічна істота

Орден фенікса

Смертежери

Симуляція протидії магічних істот

Акт.:Смертежер, маг Стрілки/Ins/Del/Esc/1/2/3/F2 Звичайний Стрілки/Ins/Del/Esc/W/A/S/D/I/J/K/L/1/2/3 Стрілки/Ins/Del/Esc/W/A/S/D/I/J/K/L/1/2/3 Стрілки/Ins/Del/Esc/W/A/S/D/I/J/K/L/1/2/3

Рисунок 2.3 – Взаємодія магічної істоти з організаціями

Приклад 3. Серіалізація та десеріалізація даних на жорсткий диск + діалогове вікно

При натисненні клавіши F2 викликається діалогове вікно (рис. 2.4), в якому користувач має обрати шлях для збереження чи зчитування даних.

Шлях до файлу «Смертежери»

Шлях до файлу «Орденовці»

Шлях до файлу «Нейтральні персонажі»

Завантажити

Зберегти

Робоча область

Симуляція протидії магічних істот

Акт.:Смертежер, маг Стрілки/Ins/Del/Esc/1/2/3/F2 Стрілки/Ins/Del/Esc/W/A/S/D/I/J/K/L/1/2/3

Рисунок 2.4 – Серіалізація даних на жорсткий диск і діалогове вікно

# 3 РОЗРОБКА ДІАГРАМ КЛАСІВ, ОБ’ЄКТІВ ТА СТАНУ

## 3.1 Діаграми класів

Для графічного подання відносин класів існують діаграми класів. Вони дають можливість розібратись у взаємовідносинах класів.

### 3.1.1 Діаграми наслідування

Рисунок 3.1 – Діаграма наслідування

Наслідування [5], зображене на рисунку 3.1, в даному програмному продукті трьохрівневе. 1-й рівень – Organization. У ньому знаходиться перелік константних значень для малювання панелей (розміри, кольори, прапорці) та функції, що реалізують серіалізацію та десеріалізацію об’єкту. Містить абстрактну функцію draw(). 2-й рівень – вдосконалює організацію методами, що дозволяють рухати панелі істот, які належать до організації. Реалізує функцію draw(). 3-й рівень – вдосконалює організацію методом, що визначає правильні координати панелі нового учасника організації.

Лістинг функцій наведено в додатку А, додатку Б, додатку В, Г

### 3.1.2 Діаграми агрегації

Організація може містити певну кількість магів, студентів та магічних істот, які можна додавати та видаляти, але це ніяким чином не вплине на роботу макрооб'єкту, так як їх існування не залежне один від одного. Таке відношення називається агрегацією (рис. 3.2).

Лістинг функцій магічної організації можна переглянути в додатку Б.

1..N

Магічна організація

Маг

Студент

Істота

Рисунок 3.2 – Діаграма агрегації

### 3.1.3 Діаграми композиції

Композиція - це відношення служить для виділення спеціальної форми відносини "частина-ціле", при якій складові частини в деякому розумінні знаходяться всередині цілого. Специфіка взаємозв'язку між ними полягає в тому, що частини не можуть виступати у відриві від цілого, тобто зі знищенням цілого знищуються і всі його складові частини (рис. 3.3).

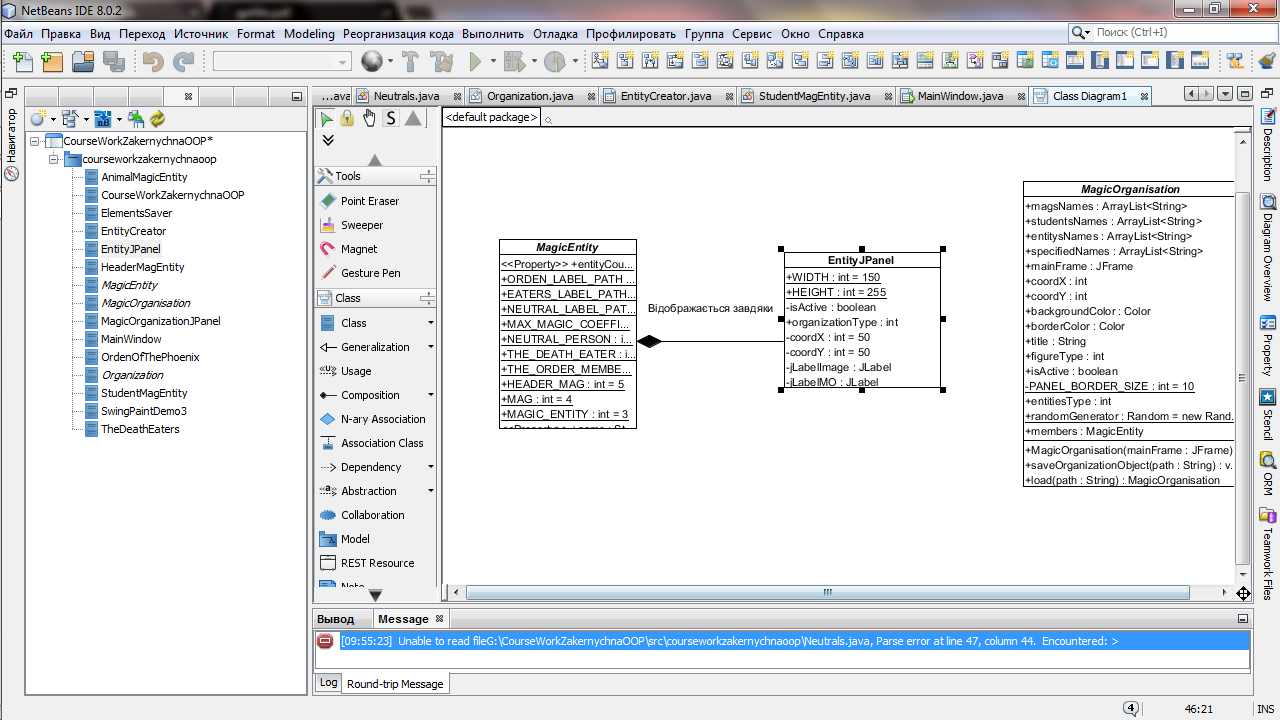


Рисунок 3.3 – Діаграма композиції

Якщо в мікрооб’єкта не була створена панель, він не буде відображений в робочій області, отже з ним неможлива буде взаємодія і з точки зору користувача його не існуватиме. При знищенні мікрооб’єкту буде знищена і його панель.

### 3.1.4 Діаграми асоціації

Відношення асоціації показує певну залежність та взаємозв’язок класів. Кількість екземплярів кожного класу, що беруть участь в асоціації, позначають на діаграмі. Приклад асоціації наведений на рисунку 3.4 [6].

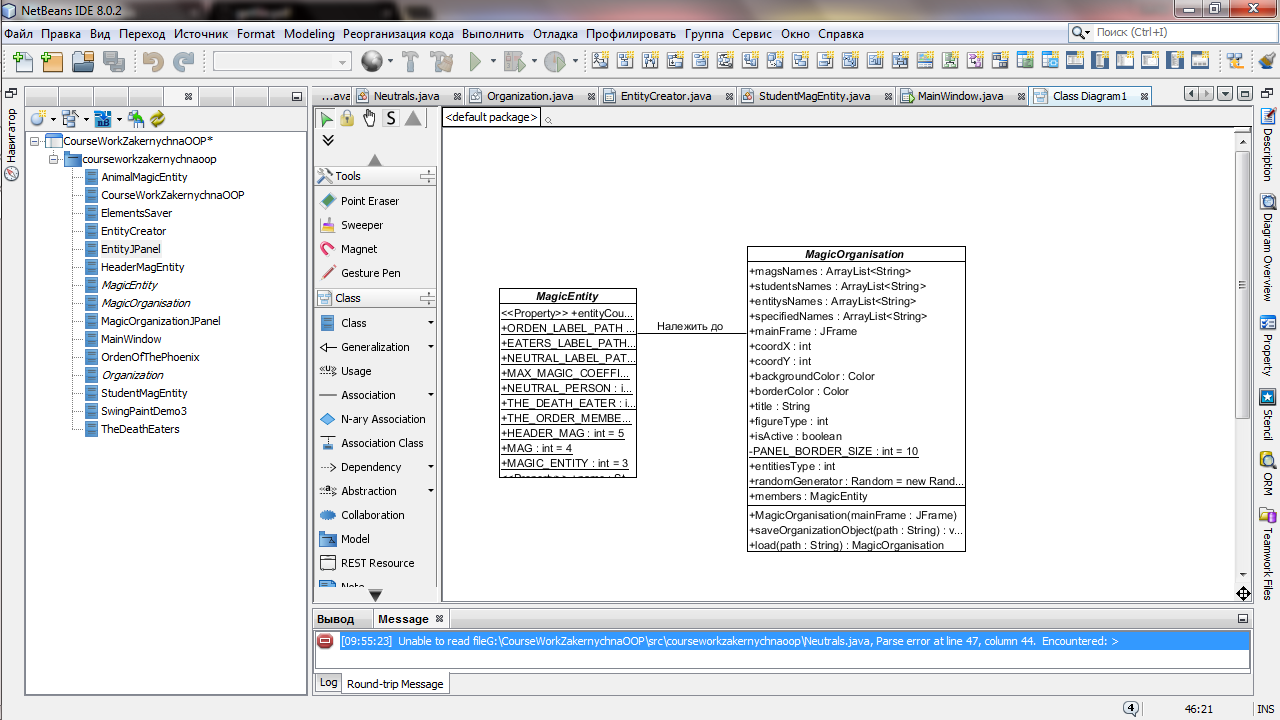
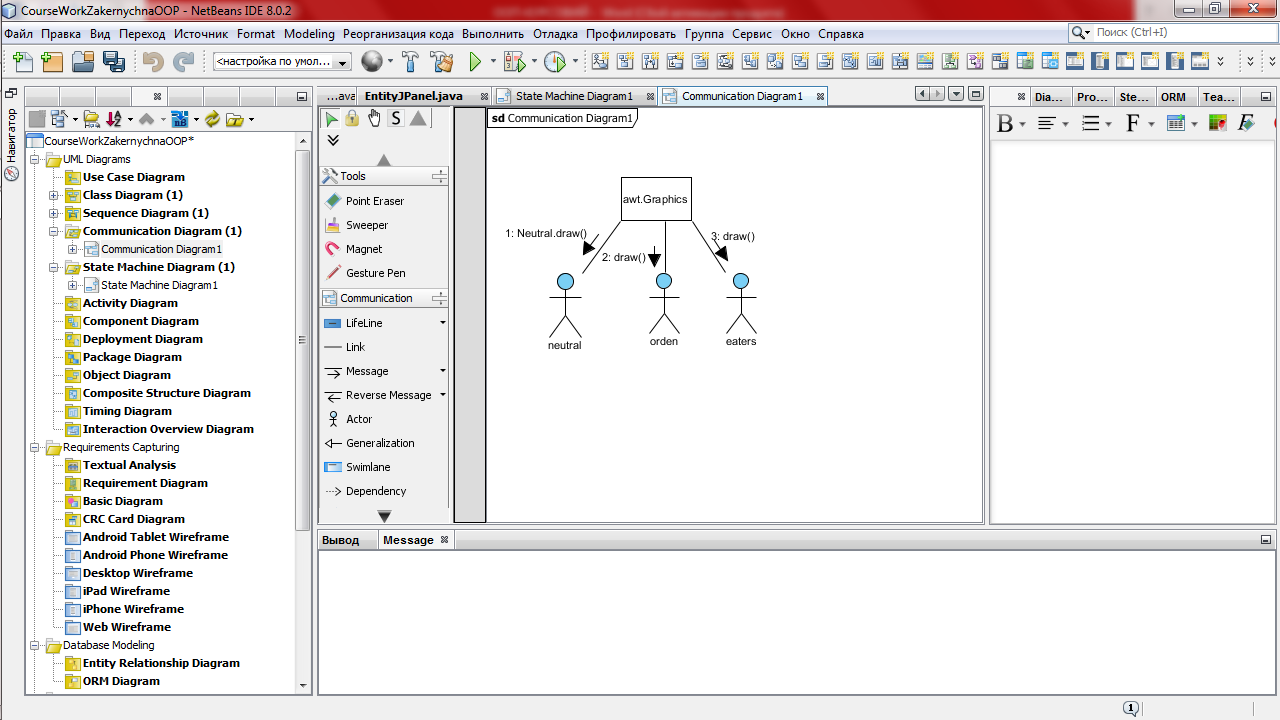


Рисунок 3.4 – Діаграма асоціації

Лише одна панель може належати магічній істоті для її відображення в робочій області, тому потужність асоціації «один до одного».

## 3.2 Діаграми кооперації

Кооперація (collaboration) - специфікація множини об'єктів окремих класів, спільно взаємодіючих з метою реалізації окремих варіантів використання в загальному контексті модельованої системи.



eaters

orden

neutrals

Рисунок 3.5 – Діаграма кооперації

Рисунок 3.5 відображає роботу функції awt.Graphics draw(), за допомогою якої рисуються макрооб’єкти програми. Лістинг даних функцій наведено в додатку К.

## 3.3. Діаграми послідовності

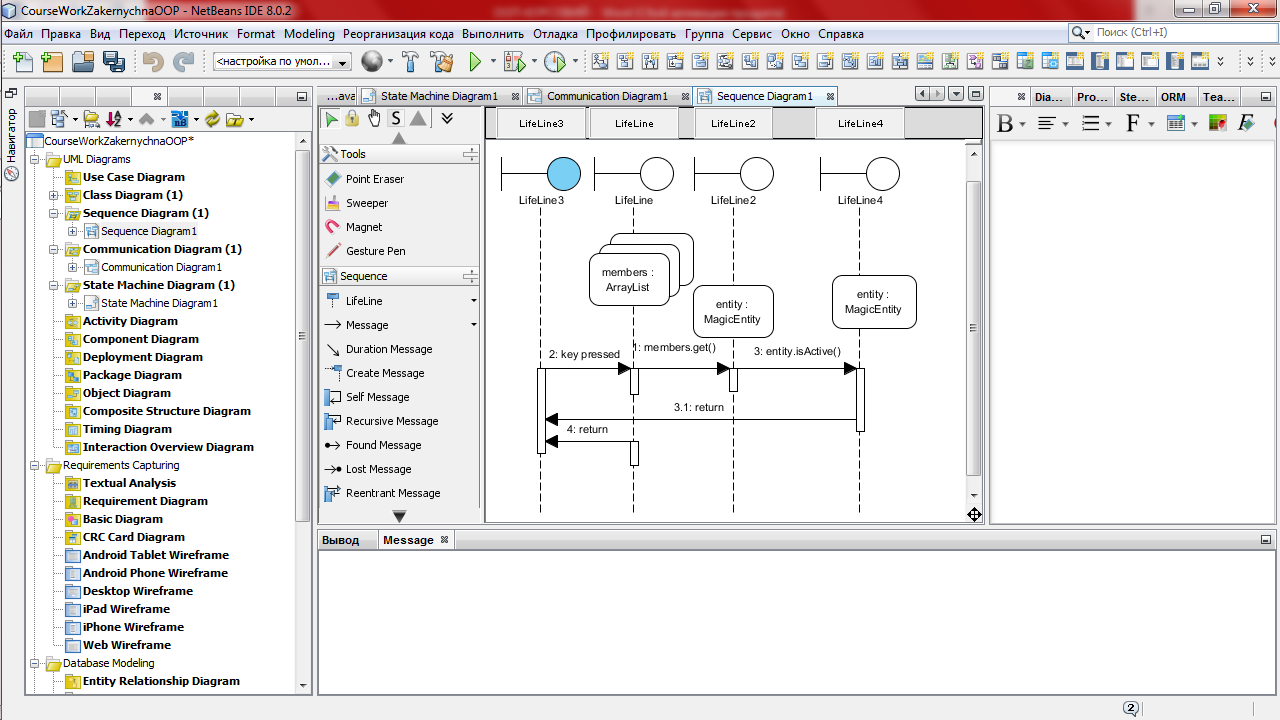


Рисунок 3.6 – Діаграма послідовності

Рисунок 3.6 демонструє переміщення активних панелей істот. Спочатку перевіряється масив істот чи є серед них активовані істоти. Якщо знайдено активовану, здійснюється її переміщення.

## 3.4 Діаграми стану

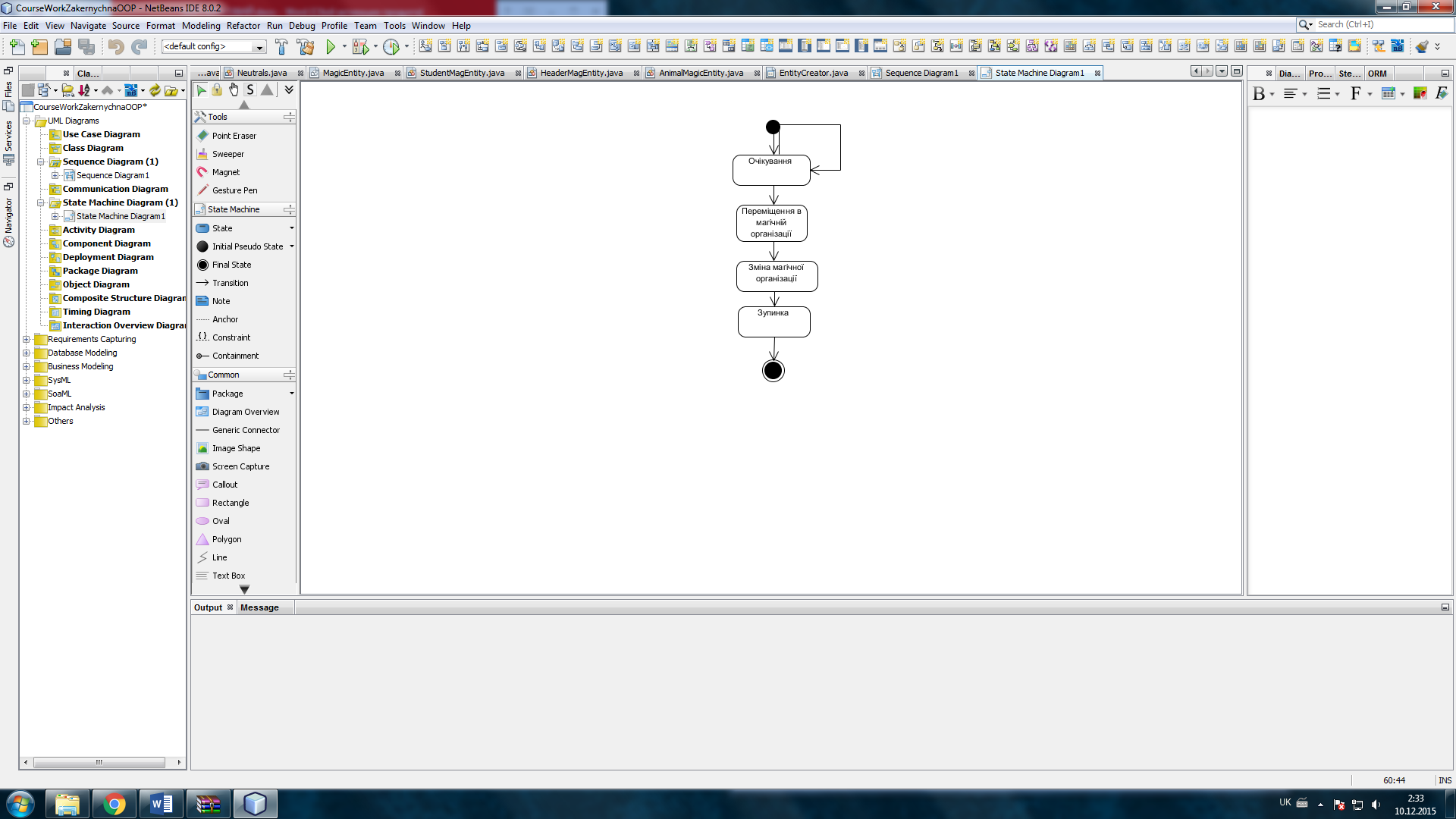


Рисунок 3.7 – Діаграма стану

Діаграма стану показує поведінку магічних істот, їх рух та реакцію на зустріч з іншим об’єктом магічного світу.

# 4 РОЗРОБКА ПІДСИСТЕМИ ГРАФІЧНОГО ВІДОБРАЖЕННЯ

## 4.1 Модель графічного відображення

Для зображення мікро- та макрооб'єктів було використано графічні примітиви та зображення.

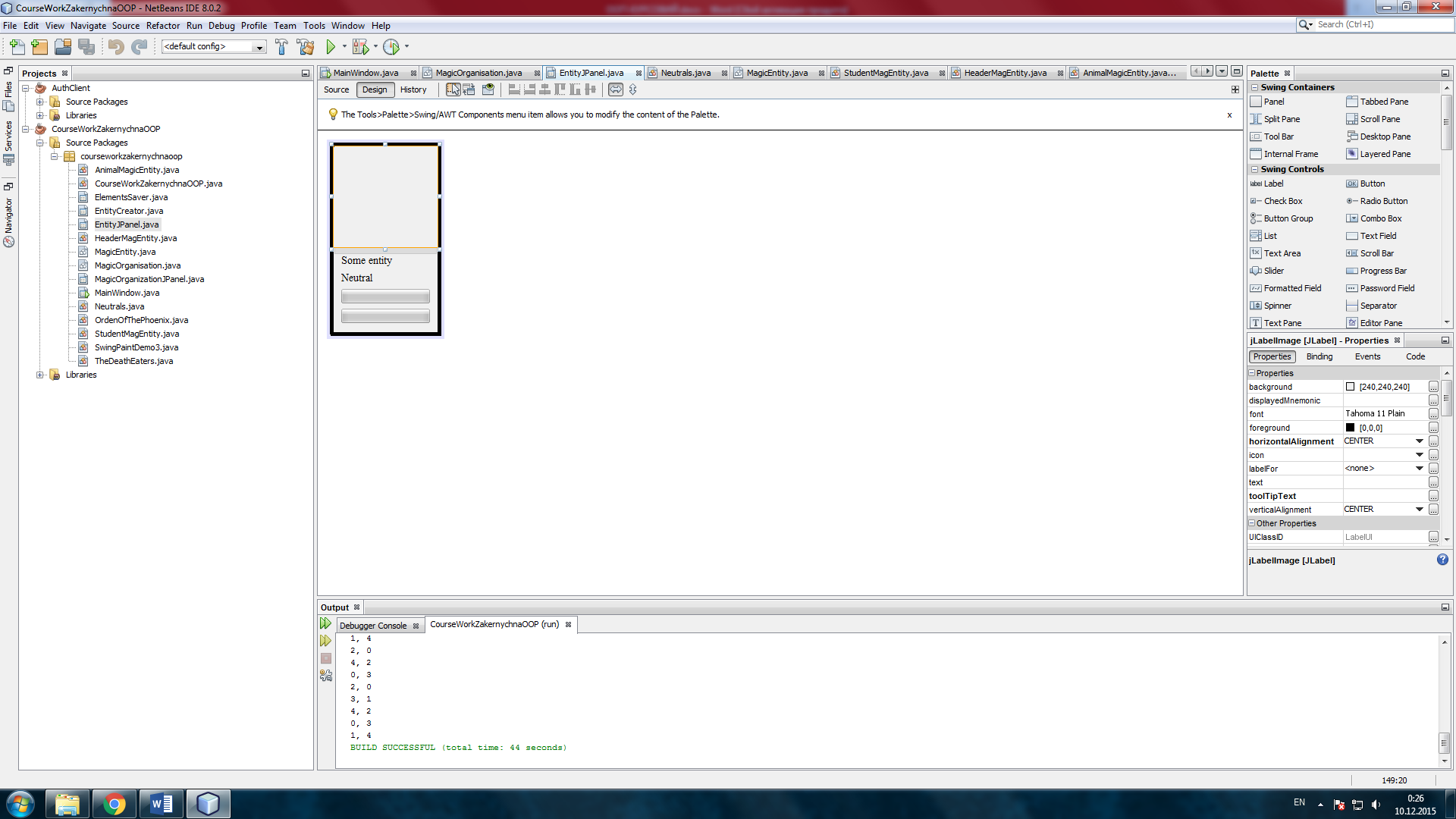


Рисунок 4.1 - Модель графічного відображення мікрооб'єкту

На рисунку 4.1 зображено модель мікрооб'єкту, що рисується за допомогою засобів Java Swing графіки, а саме: FillRectangle(), drawText() а також компонентів jLabel для відображення зображення і jProgressBar для відображення магічної сили та рівня істоти.

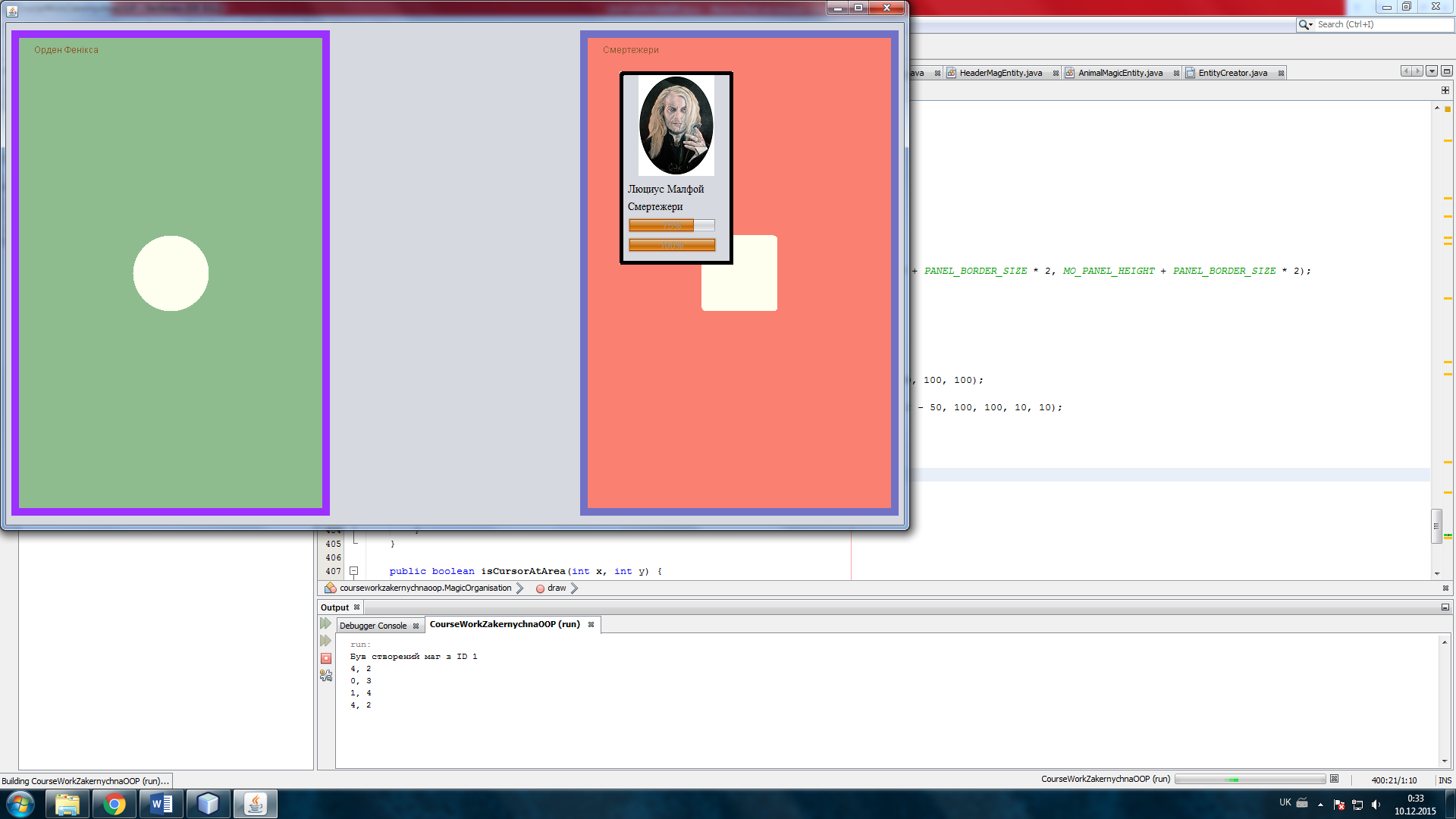


Рисунок 4.2 - Модель графічного відображення панелі макрооб’єкту

Макрооб’єкт, що показано на рисунку 4.2 рисується за допомогою засобів Java AWT графіки, а саме функцій fillRect(), fillOval(), drawString(), setColor().

## 4.2 Графічні процедури підсистеми графічного відображення

Встановлення кольору фігури здійснюється шляхом виклику методу setColor(Color color). Об’єкт класу Color має конструктор, що отримує в якості параметрів значення кольору в RGB-форматі. Усі наступні фігури будуть рисуватись обраним кольором.

Метод fillOval(int x, int y, int width, int height) дозволяє відобразити овал, у якого лівий верхній кут описаного навколо нього прямокутник буде розташований в точці (х, у), а ширина та висота будуть відповідно рівні width та height.

Прямокутник малюється за допомогою методу fillRect(int x, int y, int width, int height), що має аналогічні параметри з методом fillOval(). В результаті буде відображено прямокутник, висотою height та шириною width, лівий верхній кут якого знаходиться у точці (х, у).

Прямокутник з закругленими кутами відображається завдяки методу fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight), який аналогічний попередньому методу та додатково приймає у якості останніх параметрів розміри ребер трикутника, з якого буде обчислений кут для побудови скруглених кутів прямокутника.

Виведення тексту виконує функція drawString(String string, int x, int y), що приймає у якості параметрів текст, який потрібно відобразити та координати, починаючи з яких буде розташовуватись напис [7].

# 5 ВИКОРИСТАННЯ ЗАСОБІВ ПРОГРАМУВАННЯ JDK

## 5.1 Функції JDK для роботи з таймером.

Класи Timer і TimerTask з пакету java.util дозволяють планувати запуск завдання на певний час у майбутньому. Можна створити потік, що виконує дії у фоновому режимі і очікує заданий час. Коли час закінчиться, завдання, пов'язане з цим потоком, буде запущене. За допомогою параметрів можна запланувати завдання на повторюваний запуск або на запуск за певним часом. Не потрібно створювати потік за допомогою класу Thread, оскільки таймер спрощує цю задачу [8].

Необхідно враховувати обставину, що таймер виконується у своєму потоці і не повинен задіяти UI-елементи, які виконуються у своєму потоці. Для вирішення цієї проблеми можете використовувати метод runOnUiThread () для оновлення даних у компонентів.

Класи Timer і TimerTask працюють у зв'язці. Клас Timer використовується для планування виконання завдання. Запланована до виконання завдання має бути екземпляром класу TimerTask. Ви спочатку створюєте об'єкт класу TimerTask, а потім плануєте його запуск за допомогою класу Timer.

Клас TimerTask реалізує інтерфейс Runnabe і може бути використаний для створення потоку виконання.

У класі TimerTask мається абстрактний метод run(), який слід перевизначити. Метод повинен містити виконуваний код.

Метод cancel() перериває завдання і повертає значення true, якщо виконання завдання перервано.

Метод scheduleExecutionTime() повертає час, на який останній раз планувався запуск завдання.

Як тільки завдання створена, вона планується для виконання об'єктом класу Timer.

void cancel () - перериває потік таймера

int purge () - видаляє перервані завдання з черги таймера

void schedule (TimerTask task, long delay) - завдання task планується до виконання через період в мілісекундах, переданий в параметрі delay

void schedule (TimerTask task, long delay, long period) - завдання task планується до виконання через період в мілісекундах, переданий в параметрі delay. Потім завдання повторюється повторно періодично - кожні period мілісекунд

void schedule (TimerTask task, Date when) - завдання task планується на час, вказаний в параметрі when

void schedule (TimerTask task, Date when, long period) - завдання task планується на час, вказаний в параметрі when. Потім завдання виконується повторно періодично - кожні period мілісекунд

void scheduleAtFixedRate (TimerTask task, long delay, long period) - завдання task планується до виконання через період в мілісекундах, переданий в параметрі delay. Потім завдання виконується повторно періодично - кожні period мілісекунд. Час кожного повтору задається щодо першого запуску.

void scheduleAtFixedRate (TimerTask task, Date when, long period) - завдання task планується до виконання на час, вказаний в параметрі when. Завдання потім виконується повторно періодично - кожні period мілісекунд. Час кожного повтору задається щодо першого запуску.

Як тільки об'єкт класу Timer створений, запуск планується викликом його методу schedule ().

# 6 РОЗРОБКА ПІДСИСТЕМИ СЕРІАЛІЗАЦІЇ/ДЕСЕРІАЛІЗАЦІЇ ДАНИХ

## 6.1 Розробка формату файлу

## Серіалізация об'єктів Java дозволяє вам взяти будь-який об'єкт, який реалізує інтерфейс Serializable і перетворити його в послідовність байт, які можуть бути повністю відновлені для регенерації оригінального об'єкта [9]. Java Serialization API надає розробникам Java стандартний механізм управління сериализацией об'єктів. При збереженні об'єктів у файлі за допомогою серіалізациі використовується деякий внутрішній формат. Цей формат забезпечує збереження інформації про клас об'єкту і дозволяє відновлювати об'єкт потрібного класу, прочитаному з потоку ObjectInputStream (табл. 6.1).

Таблиця 6.1 - Формат файлу

|  |  |
| --- | --- |
| Стрічка | Опис |
| MagicOrganisation | Означає, що далі описана магічна організація. |
| magsNames | Список доступних імен персонажей-магів |
| studentsNames | Список доступних імен персонажей-студентів |
| entitysNames | Список доступних імен магічних істот |
| members; | Список MagicEntities (істот), що належать до організації |
| coordX | Координата Х панелі організації на екрані |
| coordY | Координата Y панелі організації на екрані |
| backgroundColor | Фоновий колір організації |
| borderColor | Колір рамки організації |
| title | Підпис організації |
| figureType | Тип фігури, що зображена на панелі ораганізації |
| MagicEntity | Означає, що далі описана магічна істота |
| name | Ім’я магічної істоти |
| magicPower | Магічна сила істоти |

Продовження таблиці 6.1

|  |  |
| --- | --- |
| Стрічка | Опис |
| magicLevel | Магічний рівень істоти |
| magicLevel | Магічний рівень істоти |
| entityID | ID істоти серед усіх істот світу |
| autoMove | Прапорець, чи ввімкнено автоматичний рух |
| ... | ... |

## 6.2 Механізми введення виведення мови Java

При збереженні об'єктів застосовується клас ObjectOutputStream, при відновленні - ObjectInputStream. Методи writeObject() та readObject(), які належать кожному класу відповідно, використовуються для запису в потік об’єкт або зчитування його. Так як використовується серіалізація у файл, необхідно використовувати FileOutputStream або FileInputStream, що передаються як параметри у конструктори класів ObjectOutputStream та ObjectInputStream відповідно.

Клас FileInputStream використовується для введення даних з існуючого файлу, що відкривається для читання. Об’єкти класу FileOutputStream можна незалежно від того, існує файл чи ні. При створенні нового об'єкта клас FileOutputStream перед тим, як відкрити файл для запису, спочатку створює його.

# 7 КЕРІВНИЦТВО КОРИСТУВАЧА

## 7.1 Встановлення програми та необхідні файли/каталоги

Скопіювати в цільовий каталог файл «*Magic.jar*» та каталог «images»*.* Крім цього, для роботи програми на комп’ютері повинна бути встановлена Java Virtual Machine (JVM).

## 7.2 Запуск програми

Щоб запустити програму виконайте \*.jar файл. Для коректної роботи програми забороняється видаляти каталог «images», тому що це призведе до відсутності зображень під час виконання програми.

## 7.3 Користувацький інтерфейс

Робота із програмою починається із стандартними налаштуваннями, які включають в себе два макрооб’єкти (панелі Ордена Фенікса та Смертежерів) і одного випадкового мікрооб’єкту (рис. 7.1). Магічна істота входить в ряди Смертежерів. За замовчуванням вона рухається автоматично всередині свого макрооб’єкту.

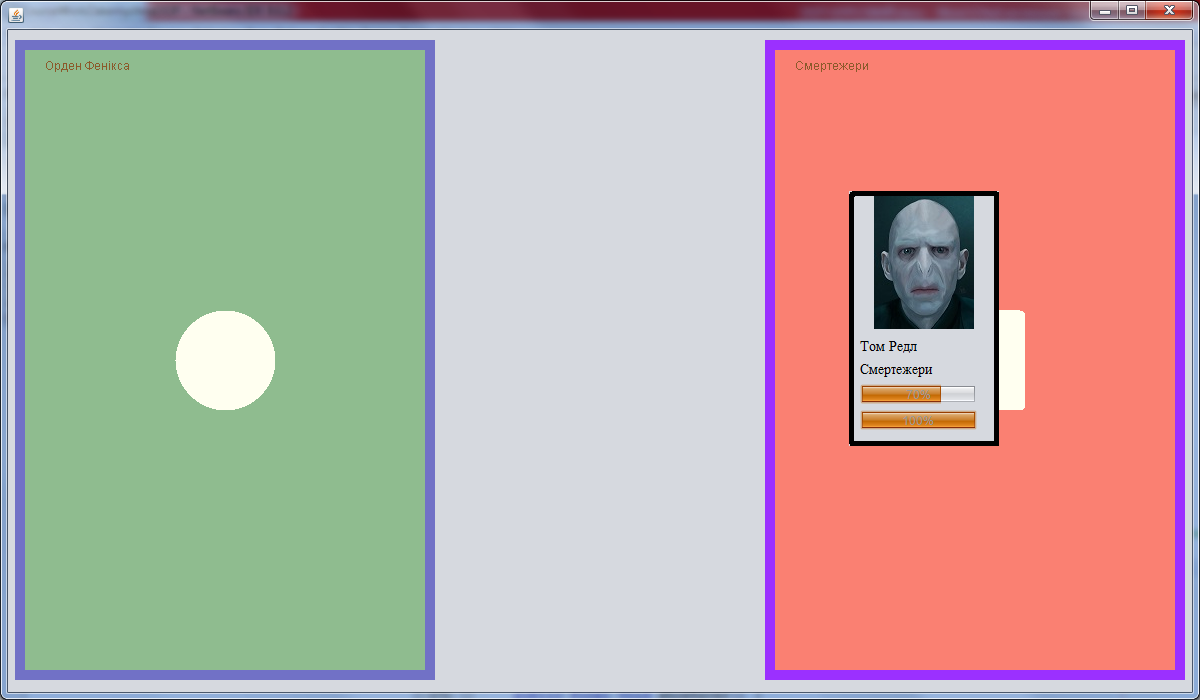


Рисунок 7.1 – Початок роботи програми

Для додавання нових істот потрібно активувати одну з панелей, після чого натиснути клавішу Insert (рис. 7.2), що відкриє вікно додавання істоти, або використати гарячі клавіші: 1 – маг, 2 – студент школи чародійства, 3 – магічне створіння.

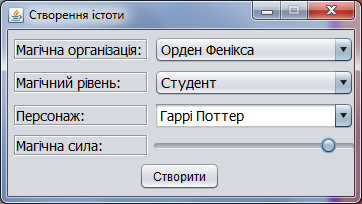


Рисунок 7.2 – Вікно створення нової істоти

Рух елементів програми можна здійснювати за допомогою клавіш Left - ліворуч, Right - праворуч, Up - вверх, Down – вниз. Для цього потрібно активувати елемент, тобто натиснути на нього лівою кнопкою миші, після чого колір рамки елемента зміниться. У істот, які належать до Ордена Фенікса рамка стане синього кольору, у Смертежерів – червоного, у нейтральних істот – жовтого (рисю 7.3). Колір рамки панелей макрооб’єктів стане фіолетового кольору. Активний макрооб’єкт рухається разом з мікрооб’єктами, які йому належать. Одночасно можна активувати декілька елементів.

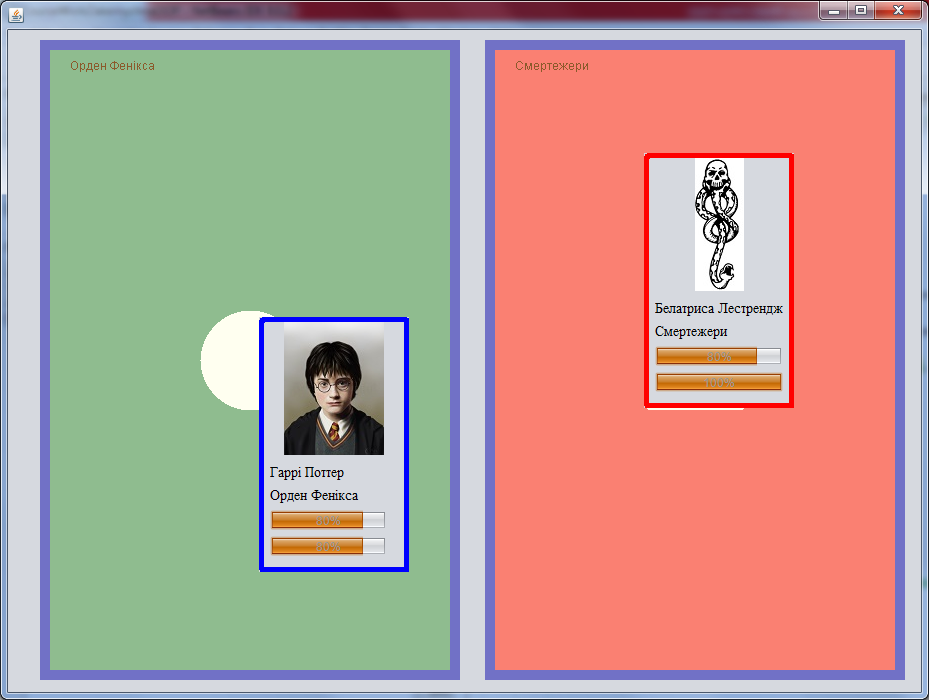


Рисунок 7.3 – Активні елементи програми

Щоб видалити елементи, необхідно виділити їх, активувати панель, до якої вони належать та натиснути клавішу Delete. Клавіша Escape деактивує всі елементи активної панелі. Клік правою кнопкою миші викликає спливаюче меню, яке дозволяє змінити належність до організації виділених об’єктів активної панелі(рис. 7.4).

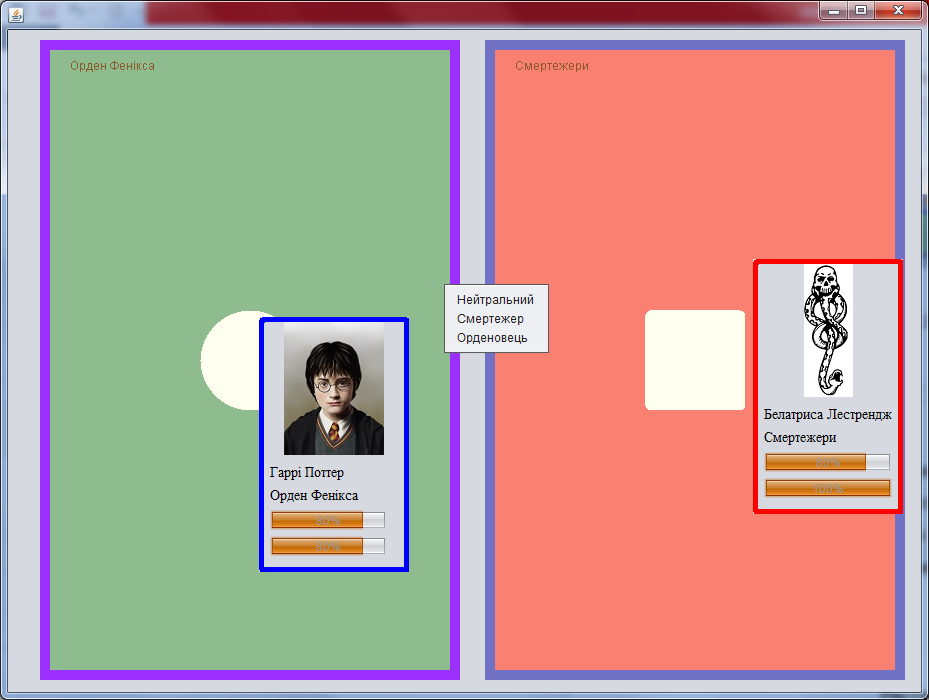


Рисунок 7.4 – Спливаюче меню

Меню збереження та відкриття файлів викликається клавішою F2, у ньому обирається шлях до файлу, у який потрібно зберегти кожен з макрооб’єктів чи завантажити його (рис. 7.5).

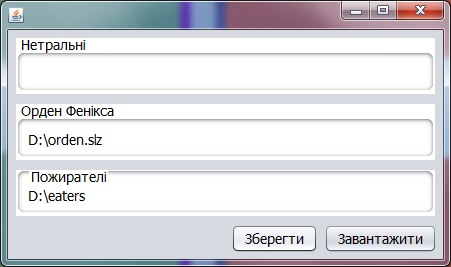


Рисунок 7.5 – Активні елементи програми

Після вибору розташування файлів (не обов’язково для усіх макрооб’єктів) потрібно натиснути кнопку «Зберегти» або «Завантажити».

# ВИСНОВКИ

Курсова робота призначена розробці програми, яка базується на основних принципах ООП, має графічний інтерфейс та забезпечує інтерактивну взаємодію користувача з елементами інтерфейсу. Першим етапом було дослідження предметної області та існуючих аналогів, розробкою технічного завдання на роботу та вибір мови програмування. Після того було розроблено структуру, графічний інтерфейс програми, UML-діаграми, серіалізацію і десеріалізацію даних та реалізовано сам продукт. Завершальним етапом стало створення керівництва користувача та прикладів використання програми.

В результаті було отримано додаток, який відповідає усім вказаним у технічному завданні вимогам та реалізує функціонал, що відсутній в аналогах – симуляція взаємодії магічних істот, що належать до різних організацій.

Також було отримано навички роботи з середовищем NetBeans та створенням графічних інтерфейсів за допомогою бібліотеки Swing.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Кей С. Хорстманн, Корнелл Г. Java. Библиотека профессионала, том 1. Основы. 9-е издание – М.: Вильямс, 2013. - 864 с.
2. Шилдт Г. Java 8. Полное руководство, 9-е издание – М.: Вильямс, 2015. – 1376 с.: ил.
3. Джеймс Гослинг, Билл Джой, Гай Стил, Гилад Брача, Алекс Бакли. Язык программирования Java SE 8. Подробное описание, 5-е издание – М.: «Вильямс», 2015. — 672 с., ил.
4. Лонг Ф., Мохиндра Д., Сикорд С., Сазерленд Ф., Свобода Д. Руководство для программиста на Java: 75 рекомендаций по написанию надежных и защищённых программ – М.: «Вильямс», 2014. — 256 с.
5. Хорстманн С. Java SE 8. Вводный — М.: «Вильямс», 2014. — 208 с.
6. Монахов Вадим. Язык программирования Java и среда NetBeans. — 3-е изд. — СПб.: БХВ-Петербург, 2011. — 704 с.
7. Барри Берд. Java 8 для чайников — М.: «Диалектика», 2015. — 400 с.
8. Брюс Эккель. Философия Java — 3-е изд. — СПб.: Питер, 2003. — 976 с.
9. Харольд Э. Р. Java ввод/вывод (Java I/O) — 4е изд. — М.: Издательство «Символ-Плюс»; 2006. — 695 стр.: ил.

# Д О Д А Т К И

# Додаток А

# Лістинг модуля Organization.java

public abstract class Organization implements Serializable{

public final static int MO\_PANEL\_WIDTH = 400;

public final static int MO\_PANEL\_HEIGHT = 620;

public final static int INTERVAL\_WIDTH = 350;

public static final Color SGI\_STATEBLUE = new Color(113, 113, 198);

public static final Color SALMON = new Color(250, 128, 114);

public static final Color DARKSEAGREEN = new Color(143, 188, 143);

public static final Color IVORY = new Color(255, 255, 240);

public static final Color TAN = new Color(139, 90, 43);

public static final Color PURPLE = new Color(155, 48, 255);

public static int FT\_OVAL = 1;

public static int FT\_ROUNDED\_RECT = 2;

public void saveOrganizationObject (String path) {

FileOutputStream fout = null;

File file = null;

try {

file = new File(path + ".slz");

if (!file.exists()) {

file.createNewFile();

}

fout = new FileOutputStream(file);

ObjectOutputStream out = new ObjectOutputStream(fout);

out.writeObject(this);

out.flush();

} catch (FileNotFoundException ex) {

Logger.getLogger(MagicOrganisation.class.getName()).log(Level.SEVERE, null, ex);

} catch (IOException ex) {

Logger.getLogger(MagicOrganisation.class.getName()).log(Level.SEVERE, null, ex);

} finally {

try {

fout.close();

} catch (IOException ex) {

Logger.getLogger(MagicOrganisation.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

public static MagicOrganisation load (String path) {

ObjectInputStream in = null;

MagicOrganisation org = null;

if(!path.endsWith(".slz")){

path += ".slz";

}

try {

in = new ObjectInputStream(new FileInputStream(path));

org =(MagicOrganisation)in.readObject();

in.close();

} catch (IOException ex) {

Logger.getLogger(MagicOrganisation.class.getName()).log(Level.SEVERE, null, ex);

} catch (ClassNotFoundException ex) {

Logger.getLogger(MagicOrganisation.class.getName()).log(Level.SEVERE, null, ex);

} finally {

try {

in.close();

} catch (IOException ex) {

Logger.getLogger(MagicOrganisation.class.getName()).log(Level.SEVERE, null, ex);

}

}

if(org != null) {

for (MagicEntity entity : org.members) {

entity.createEntityPanel();

}

}

return org;

}

public abstract void draw(Graphics g);

}

# Додаток Б

# Лістинг модуля MagicOrganization.java

public abstract class MagicOrganisation extends Organization {

public ArrayList<String> magsNames;

public ArrayList<String> studentsNames;

public ArrayList<String> entitysNames;

public ArrayList<String> specifiedNames;

public ArrayList<MagicEntity> members;

public transient JFrame mainFrame;

public int coordX;

public int coordY;

public Color backgroundColor;

public Color borderColor;

public String title;

public int figureType;

public boolean isActive;

/\*public MagicOrganisation() {

magsNames = new ArrayList<>();

entitysNames = new ArrayList<>();

studentsNames = new ArrayList<>();

specifiedNames = new ArrayList<>();

members = new ArrayList<>();

}\*/

public MagicOrganisation(JFrame mainFrame) {

this.mainFrame = mainFrame;

magsNames = new ArrayList<>();

entitysNames = new ArrayList<>();

studentsNames = new ArrayList<>();

specifiedNames = new ArrayList<>();

members = new ArrayList<>();

borderColor = SGI\_STATEBLUE;

}

public void setFrame(JFrame frame) {

mainFrame = frame;

frame.repaint();

}

public String[] getNamesList(int level) {

switch(level) {

case MagicEntity.HEADER\_MAG: return magsNames.toArray(new String[magsNames.size()]);

case MagicEntity.MAG: return studentsNames.toArray(new String[studentsNames.size()]);

case MagicEntity.MAGIC\_ENTITY: return entitysNames.toArray(new String[entitysNames.size()]);

}

return null;

}

public String getRandomMagName() {

if(magsNames.size() > 0) {

int index = randomGenerator.nextInt(magsNames.size());

return magsNames.remove(index);

}

else {

return null;

}

}

public String getRandomStudentName() {

if(studentsNames.size() > 0) {

int index = randomGenerator.nextInt(studentsNames.size());

return studentsNames.remove(index);

}

else {

return null;

}

}

public void changeMOOfPanel(int entityType) {

if(isActive){

for (MagicEntity entity : members) {

if(entity.getEntityPanel().isActivePanel()){

entity.entityType = entityType;

entity.panel.organizationType = entityType;

setNewEntityPanelLocation(entity);

entity.panel.repaint();

}

}

}

}

private void setNewEntityPanelLocation(MagicEntity entity) {

switch(entity.entityType) {

case MagicEntity.THE\_DEATH\_EATER:

entity.panel.moveAtCoords(MainWindow.eaters.coordX, MainWindow.eaters.coordY);

break;

case MagicEntity.THE\_ORDER\_MEMBER:

entity.panel.moveAtCoords(MainWindow.orden.coordX, MainWindow.orden.coordY);

break;

case MagicEntity.NEUTRAL\_PERSON:

Neutrals.setRandomCoords(entity);

break;

}

mainFrame.repaint();

}

public boolean activateEntity(int mouseX, int mouseY) {

for (MagicEntity entity : members) {

if(entity.getEntityPanel().isCursorInPanelArea(mouseX, mouseY)){

entity.getEntityPanel().changeBorderColor(true);

return true;

}

}

return false;

}

public void activatePanel () {

isActive = true;

borderColor = PURPLE;

mainFrame.repaint();

}

public void deactivatePanel() {

deactivateAllEntity();

isActive = false;

borderColor = SGI\_STATEBLUE;

mainFrame.repaint();

}

public void deactivateAllEntity(){

if(isActive){

for (MagicEntity entity : members) {

entity.getEntityPanel().changeBorderColor(false);

}

mainFrame.repaint();

}

}

public void removeAllActiveEntity() {

if(isActive){

for (MagicEntity entity : members) {

if(entity.getEntityPanel().isActivePanel()){

mainFrame.remove(entity.getEntityPanel());

//organizationPanel.remove(entity.getEntityPanel());

}

}

mainFrame.repaint();

}

}

public void addEntity(MagicEntity entity) {

members.add(entity);

switch(entity.getMagicLevel()) {

case MagicEntity.HEADER\_MAG: magsNames.remove(entity.getName());

break;

case MagicEntity.MAG: studentsNames.remove(entity.getName());

break;

case MagicEntity.MAGIC\_ENTITY: entitysNames.remove(entity.getName());

break;

}

entity.setPanelPosition(coordX, coordY);

mainFrame.add(entity.getEntityPanel());

mainFrame.repaint();

}

public void addRandomLevelEntity() {

addEntity(randomGenerator.nextInt(MagicEntity.MAGIC\_ENTITY) + MagicEntity.MAGIC\_ENTITY);

}

public void addEntity(int entityLevel) {

if(isActive) {

String name = "";

if(entityLevel == MagicEntity.HEADER\_MAG) {

name = getRandomMagName();

} else if (entityLevel == MagicEntity.MAGIC\_ENTITY){

name = getRandomEntityName();

} else if (entityLevel == MagicEntity.MAG){

name = getRandomStudentName();

}

if(name != null && !name.equals("")) {

double magicPower = ((randomGenerator.nextInt(200) + 1)/10) \* entityLevel;

MagicEntity newEntity = MagicEntity.getEntity(name, magicPower, entityLevel, entitiesType);

members.add(newEntity);

newEntity.setPanelPosition(coordX, coordY);

mainFrame.add(newEntity.getEntityPanel());

}

mainFrame.repaint();

}

}

public void addSpecifiedEntity(int entityLevel) {

throw new UnsupportedOperationException("Not supported yet.");

}

public MagicEntity getEntityAtCursorPosition(int x, int y) {

for (MagicEntity entity : members) {

if(entity.getEntityPanel().isCursorInPanelArea(x, y)){

return entity;

}

}

return null;

}

public void move (int moveX, int moveY) {

if(isActive) {

moveMOPanel(moveX, moveY);

}

else {

moveActivePanels(moveX, moveY);

}

mainFrame.repaint();

}

private void moveActivePanels(int moveX, int moveY){

for (MagicEntity entity : members) {

if(entity.getEntityPanel().isActivePanel()){

entity.getEntityPanel().movePanel(moveX, moveY, coordX, coordY);

}

}

}

private void moveMOPanel(int moveX, int moveY) {

coordX += moveX;

coordY += moveY;

if(coordX + MO\_PANEL\_WIDTH / 2 < 0 || coordX + MO\_PANEL\_WIDTH / 2 > MainWindow.WIDTH) {

coordX -= moveX;

moveX = 0;

}

if(coordY + MO\_PANEL\_HEIGHT / 2 < 0 || coordY + MO\_PANEL\_HEIGHT / 2 > MainWindow.HEIGHT) {

coordY -= moveY;

moveY = 0;

}

if(!(moveX == 0 && moveY == 0)) {

moveAllEntity(moveX, moveY);

}

}

private void moveAllEntity(int moveX, int moveY) {

for (MagicEntity entity : members) {

entity.getEntityPanel().movePanel(moveX, moveY, coordX, coordY);

}

}

public String getRandomEntityName() {

if(entitysNames.size() > 0) {

int index = randomGenerator.nextInt(entitysNames.size());

return entitysNames.remove(index);

}

else {

return null;

}

}

public boolean isPanelCoordsInMOPanel(int x, int y) {

return (coordX > x + EntityJPanel.WIDTH / 2 ||

coordX + MO\_PANEL\_WIDTH < x + EntityJPanel.WIDTH / 2) &&

(coordY > y + EntityJPanel.HEIGHT / 2 ||

coordY + MO\_PANEL\_HEIGHT < y + EntityJPanel.HEIGHT / 2);

}

public void draw(Graphics g) {

//draw borders

g.setColor(borderColor);

g.fillRect(coordX - PANEL\_BORDER\_SIZE, coordY - PANEL\_BORDER\_SIZE, MO\_PANEL\_WIDTH + PANEL\_BORDER\_SIZE \* 2, MO\_PANEL\_HEIGHT + PANEL\_BORDER\_SIZE \* 2);

//draw panel

g.setColor(backgroundColor);

g.fillRect(coordX, coordY, MO\_PANEL\_WIDTH, MO\_PANEL\_HEIGHT);

//draw figure

g.setColor(IVORY);

if(figureType == FT\_OVAL) {

g.fillOval(coordX + MO\_PANEL\_WIDTH / 2 - 50, coordY + MO\_PANEL\_HEIGHT / 2 - 50, 100, 100);

} else if (figureType == FT\_ROUNDED\_RECT){

g.fillRoundRect(coordX + MO\_PANEL\_WIDTH / 2 - 50, coordY + MO\_PANEL\_HEIGHT / 2 - 50, 100, 100, 10, 10);

}

//draw text

g.setColor(TAN);

g.drawString(title, coordX + 20, coordY + 20);

for (MagicEntity entity : members) {

entity.getEntityPanel().repaint();

}

}

public boolean isCursorAtArea(int x, int y) {

return (coordX - PANEL\_BORDER\_SIZE <= x && coordX + MO\_PANEL\_WIDTH + PANEL\_BORDER\_SIZE >= x) &&

(coordY - PANEL\_BORDER\_SIZE <= y && coordY + MO\_PANEL\_HEIGHT + PANEL\_BORDER\_SIZE >= y);

}

private static final int PANEL\_BORDER\_SIZE = 10;

public int entitiesType;

public static Random randomGenerator = new Random();

}

# Додаток B

# Лістинг модуля OrdenOfThePhoenix.java

public class OrdenOfThePhoenix extends MagicOrganisation{

private void createArrays() {

magsNames.add("Джеймс Поттер");

magsNames.add("Лілі Поттер");

magsNames.add("Артур Візлі");

magsNames.add("Моллі Візлі");

magsNames.add("Мінерва Макгонагал");

magsNames.add("Дедалус Дігл");

magsNames.add("Френк Лонгботтом");

magsNames.add("Аліса Лонгботтом");

magsNames.add("Аберфорт Дамблдор");

magsNames.add("Аластор Муді");

magsNames.add("Ремус Люпин");

magsNames.add("Сіріус Блек");

magsNames.add("Рубеус Хагрід");

studentsNames.add("Гаррі Поттер");

studentsNames.add("Герміона Грейнджер");

studentsNames.add("Рон Візлі");

studentsNames.add("Симус Фініган");

studentsNames.add("Джордж Візлі");

studentsNames.add("Фред Візлі");

studentsNames.add("Джині Візлі");

magsNames.add("Альбус Дамблдор");

magsNames.add("Пітер Петігрю");

magsNames.add("Северус Снейп");

//specifiedNames.add("Пітер Петігрю");

//specifiedNames.add("Северус Снейп");

entitysNames.add("Фенікс Фоукс");

entitysNames.add("Гіпогриф Бакбик");

entitysNames.add("Тестрал");

entitysNames.add("Кентавр Фіренце");

entitysNames.add("Домовий ельф Доббі");

}

/\*public OrdenOfThePhoenix() {

super();

createArrays();

entitiesType = MagicEntity.THE\_ORDER\_MEMBER;

organizationPanel = new MagicOrganizationJPanel("images/hogwarts.jpg");

organizationPanel.setBounds(0, 0, MO\_PANEL\_WIDTH, MO\_PANEL\_HEIGHT);

}\*/

public OrdenOfThePhoenix(JFrame frame) {

super(frame);

createArrays();

entitiesType = MagicEntity.THE\_ORDER\_MEMBER;

coordX = 25;

coordY = 50;

backgroundColor = DARKSEAGREEN;

title = "Орден Фенікса";

figureType = FT\_OVAL;

}

@Override

public void addSpecifiedEntity(int entityLevel) {

throw new UnsupportedOperationException("Not supported yet.");

}

}

# Додаток Г

# Лістинг модуля TheDeathEaters.java

public class TheDeathEaters extends MagicOrganisation{

private void createArrays() {

magsNames.add("Белатриса Лестрендж");

magsNames.add("Люциус Малфой");

studentsNames.add("Драко Малфой");

studentsNames.add("Пансі Паркінсон");

studentsNames.add("Блейз Забіні");

studentsNames.add("Грегорі Гойл");

magsNames.add("Барті Крауч");

magsNames.add("Ігорь Каркаров");

magsNames.add("Долорес Амбридж");

magsNames.add("Том Редл");

//specifiedNames.add("Ігорь Каркаров");

entitysNames.add("Змія Нагайна");

entitysNames.add("Дементор");

}

/\*public TheDeathEaters() {

super();

createArrays();

entitiesType = MagicEntity.THE\_DEATH\_EATER;

organizationPanel = new MagicOrganizationJPanel("images/forest.jpg");

organizationPanel.setBounds(MO\_PANEL\_WIDTH + 198, 0, MO\_PANEL\_WIDTH, MO\_PANEL\_HEIGHT);

}\*/

public TheDeathEaters(JFrame frame) {

super(frame);

createArrays();

entitiesType = MagicEntity.THE\_DEATH\_EATER;

coordX = MO\_PANEL\_WIDTH + 25 + INTERVAL\_WIDTH;

coordY = 50;

backgroundColor = SALMON;

title = "Смертежери";

figureType = FT\_ROUNDED\_RECT;

}

@Override

public void addSpecifiedEntity(int entityLevel) {

throw new UnsupportedOperationException("Not supported yet.");

}

}

# Додаток Д

# Лістинг модуля MagicEntity.java

public abstract class MagicEntity implements Serializable{

public static int entityCounter = 0;

public static String ORDEN\_LABEL\_PATH = "images/theOrdenOfPhoenixLabel.jpg";

public static String EATERS\_LABEL\_PATH = "images/theDeathEatersLabel.jpg";

public static String NEUTRAL\_LABEL\_PATH = "images/hatLabel.jpg";

public final static int MAX\_MAGIC\_COEFFICIENT = 20;

public final static int NEUTRAL\_PERSON = 0;

public final static int THE\_DEATH\_EATER = 1;

public final static int THE\_ORDER\_MEMBER = 2;

public final static int HEADER\_MAG = 5;

public final static int MAG = 4;

public final static int MAGIC\_ENTITY = 3;

public String name;

public double magicPower;

public int magicLevel;

public int entityType;

public int entityID;

public static Random randomGenerator;

public transient EntityJPanel panel;

public static Map entityIconPathNames = new HashMap();

public static MagicEntity getEntity(String name, double magicPower, int magicLevel, int entityType) {

MagicEntity entity = null;

switch (magicLevel) {

case HEADER\_MAG:

entity = new HeaderMagEntity(name, magicPower, entityType);

break;

case MAG:

entity = new StudentMagEntity(name, magicPower, entityType);

break;

case MAGIC\_ENTITY:

entity = new AnimalMagicEntity(name, magicPower, entityType);

break;

}

if(entity != null) entity.createEntityPanel();

return entity;

}

/\*public MagicEntity(String name, double magicPower, int magicLevel, int entityType) {

entityCounter++;

randomGenerator = new Random();

this.entityID = entityCounter;

this.name = name;

this.magicPower = magicPower;

this.magicLevel = magicLevel;

this.entityType = entityType;

createEntityPanel();

System.out.println("Була створена магічна істота з ID " + this.entityID);

}

public MagicEntity() {

this("Some entity", 1.0, 1, NEUTRAL\_PERSON);

}

public MagicEntity(MagicEntity entity) {

this(entity.getName(), entity.getMagicPower(), entity.getMagicLevel(), entity.getEntityType());

}\*/

public void createEntityPanel() {

String iconPath = NEUTRAL\_LABEL\_PATH;

if(entityIconPathNames.containsKey(name)){

iconPath = (String) entityIconPathNames.get(name);

}

else{

if (entityType == THE\_DEATH\_EATER) {

iconPath = EATERS\_LABEL\_PATH;

}

else if (entityType == THE\_ORDER\_MEMBER) {

iconPath = ORDEN\_LABEL\_PATH;

}

}

panel = new EntityJPanel(iconPath, name, magicPower, magicLevel, entityType);

}

public void setPanelPosition(int opX, int opY){

int offsetX = 0;

int offsetY = 0;

offsetX = randomGenerator.nextInt(MagicOrganisation.MO\_PANEL\_WIDTH - 147) + opX;

offsetY = randomGenerator.nextInt(MagicOrganisation.MO\_PANEL\_HEIGHT - 255) + 1;

panel.setPosition(offsetX, offsetY);

if(entityType == NEUTRAL\_PERSON) {

Neutrals.setRandomCoords(this);

}

panel.setVisible(true);

}

@Override

public String toString() {

String str = entityID + ". " + name + ":\nМагічна сила - " + magicPower +

"\nМагічний рівень - ";// + magicLevel + "\nНалежність до органіації - ";

if (magicLevel == HEADER\_MAG) str += "Могутній маг";

else if (magicLevel == MAG) str += "Маг";

else if (magicLevel == MAGIC\_ENTITY) str += "Магічна істота";

else str += "Невідомий тип магічної істоти";

str += "\nНалежність до органіації - ";

str += getEntityTypeText(entityType);

return str;

}

public static int getEntityCounter() { return entityCounter;}

public int getEntityID() { return entityID; }

public int getMagicLevel() { return magicLevel; }

public int getEntityType() { return entityType; }

public static String getEntityTypeText(int entityType) {

String str = "";

if (entityType == NEUTRAL\_PERSON) str += "Нейтральний";

else if (entityType == THE\_DEATH\_EATER) str += "Смертежери";

else if (entityType == THE\_ORDER\_MEMBER) str += "Орден Фенікса";

else str += "Невідомий тип магічної істоти";

return str;

}

public double getMagicPower() { return magicPower; }

public String getName() { return name; }

public EntityJPanel getEntityPanel() { return panel; }

public void setMagicLevel(int magicLevel) { this.magicLevel = magicLevel; }

public void setEntityType(int entityType) { this.entityType = entityType; }

public void setMagicPower(double magicPower) { this.magicPower = magicPower; }

public void setName(String name) { this.name = name; }

public void setEntityPanel(EntityJPanel panel) { this.panel = panel; }

static void addIconNames() {

entityIconPathNames.put("Северус Снейп", "images/Severus.jpg");

entityIconPathNames.put("Альбус Дамблдор", "images/Dumbledore.jpg");

entityIconPathNames.put("Гаррі Поттер", "images/Harry.jpg");

entityIconPathNames.put("Люциус Малфой", "images/Lucius.jpg");

entityIconPathNames.put("Сіріус Блек", "images/Sirius.jpg");

entityIconPathNames.put("Том Редл", "images/Voldemort.jpg");

}

static {

addIconNames();

}

}

# Додаток Е

# Лістинг модуля HeaderMagEntity.java

public class HeaderMagEntity extends MagicEntity{

public final static int MAGIC\_LEVEL = HEADER\_MAG;

public HeaderMagEntity(String name, double magicPower, int entityType) {

entityCounter++;

randomGenerator = new Random();

this.magicLevel = MAGIC\_LEVEL;

this.entityID = entityCounter;

this.name = name;

this.magicPower = magicPower <= MAX\_MAGIC\_POWER ? magicPower : MAX\_MAGIC\_POWER;

this.entityType = entityType;

createEntityPanel();

System.out.println("Був створений маг з ID " + this.entityID);

}

public HeaderMagEntity() {

this("Some entity", 1.0, NEUTRAL\_PERSON);

}

public HeaderMagEntity(MagicEntity entity) {

this(entity.getName(), entity.getMagicPower(), entity.getEntityType());

}

private final static int MAX\_MAGIC\_POWER = MAX\_MAGIC\_COEFFICIENT \* MAGIC\_LEVEL;

}

# Додаток Ж

# Лістинг модуля StudentMagEntity.java

public class StudentMagEntity extends MagicEntity {

public final static int MAGIC\_LEVEL = MAG;

public StudentMagEntity(String name, double magicPower, int entityType) {

entityCounter++;

randomGenerator = new Random();

this.magicLevel = MAGIC\_LEVEL;

this.entityID = entityCounter;

this.name = name;

this.magicPower = magicPower <= MAX\_MAGIC\_POWER ? magicPower : MAX\_MAGIC\_POWER;

this.entityType = entityType;

createEntityPanel();

System.out.println("Був створений студент школи чаклунства з ID " + this.entityID);

}

public StudentMagEntity() {

this("Some entity", 1.0, NEUTRAL\_PERSON);

}

public StudentMagEntity(MagicEntity entity) {

this(entity.getName(), entity.getMagicPower(), entity.getEntityType());

}

private final static int MAX\_MAGIC\_POWER = MAX\_MAGIC\_COEFFICIENT \* MAGIC\_LEVEL;

}

# Додаток З

# Лістинг модуля AnimalMagicEntity.java

public class AnimalMagicEntity extends MagicEntity {

public final static int MAGIC\_LEVEL = MAGIC\_ENTITY;

public AnimalMagicEntity(String name, double magicPower, int entityType) {

entityCounter++;

randomGenerator = new Random();

this.magicLevel = MAGIC\_LEVEL;

this.entityID = entityCounter;

this.name = name;

this.magicPower = magicPower <= MAX\_MAGIC\_POWER ? magicPower : MAX\_MAGIC\_POWER;

this.entityType = entityType;

createEntityPanel();

System.out.println("Була створена магічна істота з ID " + this.entityID);

}

public AnimalMagicEntity() {

this("Some entity", 1.0, NEUTRAL\_PERSON);

}

public AnimalMagicEntity(MagicEntity entity) {

this(entity.getName(), entity.getMagicPower(), entity.getEntityType());

}

private final static int MAX\_MAGIC\_POWER = MAX\_MAGIC\_COEFFICIENT \* MAGIC\_LEVEL;

# Додаток И

# Лістинг модуля EntityJPanel.java

public class EntityJPanel extends javax.swing.JPanel {

public final static int WIDTH = 150;

public final static int HEIGHT = 255;

private boolean isActive;

public int organizationType;

private int coordX = 50;

private int coordY = 50;

public EntityJPanel(String fileName, String name, double magicPower, int level, int organizationType) {

initComponents();

isActive = false;

this.organizationType = organizationType;

jLabelName.setText(name);

jLabelMO.setText(MagicEntity.getEntityTypeText(organizationType));

loadImage(fileName);

createProgressBars();

jProgressBarLevel.setValue(level);

jProgressBarMagicPower.setValue((int) magicPower);

/\*addMouseListener(new MouseAdapter() {

@Override

public void mousePressed(MouseEvent e) {

if (e.isPopupTrigger()) {

menu.show(e.getComponent(), e.getX(), e.getY());

}

}

});

addMouseMotionListener(new MouseAdapter() {

@Override

public void mouseDragged(MouseEvent e) {

//System.out.println("drag" + e.getX() + ", " + e.getY());

moveAtCoords(e.getX(),e.getY());

}

}); \*/

}

private void createProgressBars() {

jProgressBarLevel.setMinimum(0);

jProgressBarLevel.setMaximum(MagicEntity.HEADER\_MAG);

jProgressBarLevel.setStringPainted(true);

jProgressBarMagicPower.setStringPainted(true);

}

public void changeBorderColor(boolean isActive) {

this.isActive = isActive;

int r = 0;

int g = 0;

int b = 0;

if(isActive){

switch(organizationType) {

case MagicEntity.NEUTRAL\_PERSON:

r = 255;

g = 255;

break;

case MagicEntity.THE\_DEATH\_EATER:

r = 255;

break;

case MagicEntity.THE\_ORDER\_MEMBER:

b = 255;

break;

}

}

setBorder(new LineBorder(new java.awt.Color(r, g, b), 5, true));

}

public void moveAtCoords(int x, int y) {

if ((coordX!=x) || (coordY!=y)) {

coordX=x;

coordY=y;

setBounds(coordX, coordY, WIDTH, HEIGHT);

}

}

@Override

public int getX (){ return coordX; }

@Override

public int getY (){ return coordY; }

public void movePanel(int x, int y, int opX, int opY) {

coordX += x;

coordY += y;

//System.out.println("Рух панелі істоти " + coordX);

if(organizationType != MagicEntity.NEUTRAL\_PERSON) {

if(coordX < opX){

coordX = opX;

}

else if(coordX > opX + MagicOrganisation.MO\_PANEL\_WIDTH - WIDTH){

coordX = opX + MagicOrganisation.MO\_PANEL\_WIDTH - WIDTH;

}

if(coordY < opY){

coordY = opY;

}

else if(coordY > opY + MagicOrganisation.MO\_PANEL\_HEIGHT - HEIGHT){

coordY = opY + MagicOrganisation.MO\_PANEL\_HEIGHT - HEIGHT;

}

setBounds(coordX, coordY, WIDTH, HEIGHT);

}

else {

if (MainWindow.orden.isPanelCoordsInMOPanel(coordX, coordY) ||

MainWindow.eaters.isPanelCoordsInMOPanel(coordX, coordY)) {

coordX -= x;

coordY -= y;

}

}

}

/\*@Override

public Dimension getPreferredSize() {

return new Dimension(150,245);

} \*/

public static int showOkCancelDialog(String theMessage) {

int result = JOptionPane.showConfirmDialog((Component) null, theMessage,

"alert", JOptionPane.OK\_CANCEL\_OPTION);

return result;

}

public boolean isCursorInPanelArea(int x, int y) {

return (x >= coordX && x <= (coordX + WIDTH)) && (y >= coordY && y <= (coordY + HEIGHT));

}

public boolean isActivePanel() {

return isActive;

}

public void setPosition(int x, int y) {

coordX = x;

coordY = y;

setBounds(coordX, coordY, WIDTH, HEIGHT);

}

private void loadImage(String fileName) {

jLabelImage.setIcon(new ImageIcon(fileName));

}

# Додаток Й

# Лістинг модуля MainWindow.java

public static OrdenOfThePhoenix orden;

public static TheDeathEaters eaters;

public static Neutrals neutrals;

private Timer timer;

public static final int WIDTH = 1200;

public static final int HEIGHT = 700;

public MainWindow() {

initComponents();

createPopupMenu();

initTimer();

setLayout(null);

orden = new OrdenOfThePhoenix(this);

eaters = new TheDeathEaters(this);

neutrals = new Neutrals(this);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

addKeyListener(new KeyAdapter() {

@Override

public void keyPressed(KeyEvent event) {

int keyID = event.getKeyCode();

switch(keyID) {

case KeyEvent.VK\_LEFT:

orden.move(-5, 0);

eaters.move(-5, 0);

neutrals.move(-5, 0);

//orden.moveActivePanels(-2, 0);

break;

case KeyEvent.VK\_RIGHT:

orden.move(5, 0);

eaters.move(5, 0);

neutrals.move(5, 0);

break;

case KeyEvent.VK\_UP:

orden.move(0, -5);

eaters.move(0, -5);

neutrals.move(0, -5);

break;

case KeyEvent.VK\_DOWN:

orden.move(0, 5);

eaters.move(0, 5);

neutrals.move(0, 5);

break;

case KeyEvent.VK\_ESCAPE:

orden.deactivateAllEntity();

eaters.deactivateAllEntity();

neutrals.deactivateAllEntity();

break;

case KeyEvent.VK\_INSERT:

new EntityCreator().setVisible(true);

//orden.addRandomLevelEntity();

//eaters.addRandomLevelEntity();

break;

case KeyEvent.VK\_DELETE:

orden.removeAllActiveEntity();

eaters.removeAllActiveEntity();

neutrals.removeAllActiveEntity();

//orden.organizationPanel.repaint();

break;

case KeyEvent.VK\_1:

orden.addEntity(MagicEntity.HEADER\_MAG);

eaters.addEntity(MagicEntity.HEADER\_MAG);

neutrals.addEntity(MagicEntity.HEADER\_MAG);

break;

case KeyEvent.VK\_2:

orden.addEntity(MagicEntity.MAG);

eaters.addEntity(MagicEntity.MAG);

neutrals.addEntity(MagicEntity.MAG);

break;

case KeyEvent.VK\_3:

orden.addEntity(MagicEntity.MAGIC\_ENTITY);

eaters.addEntity(MagicEntity.MAGIC\_ENTITY);

neutrals.addEntity(MagicEntity.MAGIC\_ENTITY);

break;

case KeyEvent.VK\_F2:

showSaver();

break;

}

}

});

addMouseListener(new MouseAdapter() {

@Override

public void mousePressed(MouseEvent e) {

if(e.getButton() == MouseEvent.BUTTON3) {

jPopupMenu1.setLocation(e.getX(), e.getY());

jPopupMenu1.setVisible(true);

} else if(e.getButton() == MouseEvent.BUTTON1) {

if(orden.isCursorAtArea(e.getX(), e.getY())) {

if (!orden.activateEntity(e.getX(), e.getY())) {

orden.activatePanel();

eaters.deactivatePanel();

}

else {

orden.deactivatePanel();

}

}

else if(eaters.isCursorAtArea(e.getX(), e.getY())) {

if (!eaters.activateEntity(e.getX(), e.getY())) {

eaters.activatePanel();

orden.deactivatePanel();

}

else {

eaters.deactivatePanel();

}

}

else {

neutrals.activateEntity(e.getX(), e.getY());

eaters.deactivatePanel();

orden.deactivatePanel();

}

}

}

});

eaters.activatePanel();

eaters.addEntity(MagicEntity.HEADER\_MAG);

setSize(WIDTH, HEIGHT);

setVisible(true);

}

private void initTimer() {

timer = new Timer(1000, new ActionListener() {

@Override

public void actionPerformed(ActionEvent ae) {

int x = (int) (System.currentTimeMillis() % 5);

int y = (int) ((System.currentTimeMillis() + 8) % 5);

System.out.println(x +", " + y);

eaters.members.get(0).getEntityPanel().movePanel(x, y, eaters.coordX, eaters.coordY);

repaint();

}});

timer.start();

}

public final void showSaver() {

new ElementsSaver(this).setVisible(true);

}

private void createPopupMenu() {

jPopupMenu1.setName("Зміна організації");

ActionListener al = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

System.out.println("pressed");

String itemText = ((JMenuItem) e.getSource()).getText();

switch (itemText) {

case EATERS\_PM\_ITEM\_LABEL:

orden.changeMOOfPanel(MagicEntity.THE\_DEATH\_EATER);

eaters.changeMOOfPanel(MagicEntity.THE\_DEATH\_EATER);

break;

case ORDEN\_PM\_ITEM\_LABEL:

orden.changeMOOfPanel(MagicEntity.THE\_ORDER\_MEMBER);

eaters.changeMOOfPanel(MagicEntity.THE\_ORDER\_MEMBER);

break;

case NEUTRAL\_PM\_ITEM\_LABEL:

orden.changeMOOfPanel(MagicEntity.NEUTRAL\_PERSON);

eaters.changeMOOfPanel(MagicEntity.NEUTRAL\_PERSON);

break; }

jPopupMenu1.setVisible(false);

}

};

JMenuItem m = new JMenuItem(NEUTRAL\_PM\_ITEM\_LABEL);

m.addActionListener(al);

jPopupMenu1.add(m);

m = new JMenuItem(EATERS\_PM\_ITEM\_LABEL);

m.addActionListener(al);

jPopupMenu1.add(m);

m = new JMenuItem(ORDEN\_PM\_ITEM\_LABEL);

m.addActionListener(al);

jPopupMenu1.add(m);

}

@Override

public void paint(Graphics g) {

super.paintComponents(g);

orden.draw(g);

eaters.draw(g);

neutrals.draw(g);

}

# Додаток К

# Лістинг модуля EntityCreator.java

public class EntityCreator extends javax.swing.JFrame {

private MagicOrganisation org;

private String entityName;

private int magicLevel;

private int magicPower;

private int entityType;

public EntityCreator() {

initComponents();

refreshCreator(); }

private void refreshCreator() {

String organization = (String)jComboBoxMO.getSelectedItem();

String magicLevel = (String)jComboBoxLevel.getSelectedItem();

this.magicLevel = -1;

switch(magicLevel) {

case "Маг": this.magicLevel = MagicEntity.HEADER\_MAG; break;

case "Студент": this.magicLevel = MagicEntity.MAG; break;

case "Істота": this.magicLevel = MagicEntity.MAGIC\_ENTITY; break; }

org = null;

switch(organization) {

case "Нейтральний":

entityType = MagicEntity.NEUTRAL\_PERSON;

org = MainWindow.neutrals;

break;

case "Орден Фенікса":

entityType = MagicEntity.THE\_ORDER\_MEMBER;

org = MainWindow.orden;

break;

case "Смертежери":

entityType = MagicEntity.THE\_DEATH\_EATER;

org = MainWindow.eaters;

break; }

if(org != null) setComboBoxNamesItems(org.getNamesList(this.magicLevel));

jSliderPower.setMinimum((this.magicLevel - 1) \* 20);

jSliderPower.setMaximum(this.magicLevel \* 20);

}

private void setComboBoxNamesItems(String[] itemList) {

if(itemList != null){

jComboBoxName.setModel(new DefaultComboBoxModel(itemList));

}

//jComboBoxName.removeItemAt(itemList.length - 1);

}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

entityName = (String) jComboBoxName.getSelectedItem();

magicPower = jSliderPower.getValue();

org.addEntity(MagicEntity.getEntity(entityName, magicPower, magicLevel, entityType));

dispose();

}

private void jComboBoxMOItemStateChanged(java.awt.event.ItemEvent evt) {

refreshCreator();

}

private void jComboBoxLevelItemStateChanged(java.awt.event.ItemEvent evt) {

refreshCreator(); }