МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра

автоматизованих систем обробки інформації та управління

**КУРСОВА РОБОТА**

з дисципліни

“ Об'єктно орієнтоване програмування ”

на тему

"Використання концепції ООП для створення гри

на мові програмування Python"

|  |  |  |
| --- | --- | --- |
| Керівник : доц. Муха І.П |  | Виконав : Кушка М. О. |
| Допущений до захисту |  | студент гр. ІП-61, ФІОТ |
| І\_\_\_І \_\_\_\_\_\_\_\_ 2017 \_\_\_\_\_\_  підпис |  | 2 курс  № IП-6116 |
| Захистив з оцінкою  \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_  оцінка підпис  І\_\_\_І \_\_\_\_\_\_\_\_\_ 2017 |  | Підпис: |

Київ 2017

Національний технічний університет України “КПІ”

(назва вищого навчального закладу)

Кафедра автоматизованих систем обробки інформації і управління

Дисципліна Основи програмування

Напрям "Програмна інженерія"

Курс 2 Група ІП-61 Семестр 1

**ЗАВДАННЯ**

**на курсову роботу студента**

|  |
| --- |
| Кушки Михайла Олександровича |

(прізвище, ім’я, по батькові)

|  |  |
| --- | --- |
| 1. Тема роботи | Використання концепції ООП для створення гри на мові |
| програмування Python | |
|  | |

|  |  |
| --- | --- |
| 2. Строк здачі студентом закінченої роботи |  |

|  |  |
| --- | --- |
| 3. Вихідні дані до роботи | Оскільки дана робота є написанням гри, то жодних вхідних даних |
| для старту програми не передбачається | |
|  | |
|  | |

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

|  |
| --- |
|  |
|  |
|  |
|  |

5. Перелік графічного матеріалу ( з точним зазначенням обов’язкових креслень )

|  |
| --- |
|  |
|  |
|  |
|  |

|  |  |
| --- | --- |
| 6. Дата видачі завдання |  |

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Підписи керівника, студента |
| 1. | Отримання теми курсової роботи |  |  |
| 2. | Підготовка ТЗ |  |  |
| 3. | Пошук та вивчення літератури з питань курсової роботи |  |  |
| 4. | Розробка алгоритму вирішення задачі |  |  |
| 6. | Узгодження алгоритму з керівником |  |  |
| 5. | Розробка сценарію роботи програми |  |  |
| 6. | Узгодження сценарію роботи програми з керівником |  |  |
| 7. | Узгодження з керівником інтерфейсу користувача |  |  |
| 8. | Розробка програмного забезпечення |  |  |
| 9. | Налагодження розрахункової частини програми |  |  |
| 10. | Розробка та налагодження інтерфейсної частини програми |  |  |
| 11. | Узгодження з керівником набору тестів для контрольного прикладу |  |  |
| 12. | Тестування програми |  |  |
| 13. | Підготовка пояснювальної записки |  |  |
| 14. | Здача курсової роботи на перевірку |  |  |
| 15. | Захист курсової роботи |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  |  | Кушка М. О. |
|  | (підпис) |  | (прізвище, ім’я, по батькові) |

|  |  |  |  |
| --- | --- | --- | --- |
| Керівник |  |  | Муха І. П. |
|  | (підпис) |  | (прізвище, ім’я, по батькові) |

"\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ р.

# Анотація

Пояснювальна записка до курсової роботи: 41 сторінка, 4 рисунки, 2 таблиці, 4 посилання.

Об’єкт дослідження: концепція ООП в мові програмування Python

Мета роботи: розробити гру на мові програмування Python, представивши основні ігрові об’єкти і компоненти у вигляді класів. Організувати наслідування та взаємодію класів між собою для полегшення розробки та підтримки коду даної програми.

В розділі “Теоретичні викладки” описано основні теоретичні засади для написання гри, а також її жанр, тип, особливості.

В “описі алгоритмів” був описаний основний алгоритм ігрового процесу та деякі допоміжні алгоритми, що уточнюють деякі пункти з основного.

“Опис програмного забезпечення” дає детальну інформацію щодо класів, присутніх в коді основної та допоміжних програм, їх методів та деталі щодо використання у програмі; зображена діаграма класів для усієї ієрархії класів.

Розділ “Тестування програмного забезпечення” розкриває основні аспекти реалізації суттєві з точки зору користувача, а саме: при натисканні клавіші “пробіл” корабель випускає ракету тощо. Також були перевірені комбінації, що теоретично можуть призвести до некоректної чи нелогічної поведінки програми.

В “інструкції користувача” подані основна суть гри та інструкція щодо гри, а саме як нею користуватися, тобто грати.

Національний технічний університет України

“Київський політехнічний інститут”

Кафедра автоматизованих систем обробки інформації та управління

|  |  |  |
| --- | --- | --- |
| **РОЗРОБИВ** |  | **ЗАТВЕРДИВ** |
| **Студент**  \_\_\_\_\_\_\_\_\_\_*Кушка М. О.*  (підпис) (ініціали, прізвище)  “ ” 2017 р. |  | **Керівник**  \_\_\_\_\_\_\_\_\_\_ *І.П.Муха*  (підпис) (ініціали, прізвище)  “ ” 2017 р. |

Використання концепції ООП для створення гри на мові

програмування Python

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

з дисципліни:

«Об'єктно-орієнтоване програмування»

Шифр КПІ ІП-6116.ТЗ

на 14 сторінках

Київ 2017

ЗМІСТ ТЗ

[1 Найменування та область застосування 8](#_Toc502185387)

[2.1 Перелік документів, на підставі яких ведеться розробка 9](#_Toc502185388)

[2.2 Найменування і умовне позначення теми розробки 9](#_Toc502185389)

[3 Призначення розробки 10](#_Toc502185390)

[3.1 Функціональне призначення розробки 10](#_Toc502185391)

[3.2 Експлуатаційне призначення 10](#_Toc502185392)

[4 Вимоги до програмного забезпечення 11](#_Toc502185393)

[4.1 Вимоги до функціональних характеристик 11](#_Toc502185394)

[4.1.1 Вимоги до складу виконуваних функцій 11](#_Toc502185395)

[4.1.2 Вимоги до організації вхідних даних 11](#_Toc502185396)

[4.1.3 Вимоги до організації вихідних даних 11](#_Toc502185397)

[4.2 Вимоги до надійності 11](#_Toc502185398)

[4.2.1 Вимоги до надійного функціонування програми 11](#_Toc502185399)

[4.2.2 Час відновлення після відмови 12](#_Toc502185400)

[4.2.3 Відмови через некоректні дії користувача 12](#_Toc502185401)

[4.3 Умови експлуатації 12](#_Toc502185402)

[4.3.1 Кліматичні умови експлуатації 12](#_Toc502185403)

[4.3.2 Вимоги до видів обслуговування 12](#_Toc502185404)

[4.3.3 Вимоги до кількості та кваліфікації персоналу 12](#_Toc502185405)

[4.4 Вимоги до складу і параметрів технічних засобів 13](#_Toc502185406)

[4.5 Вимоги до інформаційної і програмної сумісності 13](#_Toc502185407)

[4.6 Вимоги до маркування та упаковки 13](#_Toc502185408)

[4.7 Вимоги до транспортування та зберігання 13](#_Toc502185409)

[5 Вимоги до програмної документації 14](#_Toc502185410)

[5.1 Попередній склад програмної документації 14](#_Toc502185411)

[6 Техніко-економічні показники 15](#_Toc502185412)

[7 Стадії та етапи розробки 16](#_Toc502185413)

[7.1 Стадії розробки 16](#_Toc502185414)

[7.2 Етапи розробки 16](#_Toc502185415)

[7.3 Зміст роботи по етапам 16](#_Toc502185416)

[8 Порядок контролю і прийому 18](#_Toc502185417)

[8.1 Види досліджень 18](#_Toc502185418)

[8.2 Загальні вимоги до прийняття роботи 18](#_Toc502185419)

1. Найменування та область застосування

Дана робота присвячена розробці програмного забезпечення для демонстрації основних концепцій ООП через написання гри на мові програмування Python. Вихідний код даного програмного забезпечення може використовуватися для демонстрації реалізації великого проекту з використанням ООП.

# Підстави для розробки

* 1. Перелік документів, на підставі яких ведеться розробка

Підставою для проведення розробки являється наказ № 6 від 11 травня 2016 року. Наказ затверджено на засіданні кафедри АСОІУ, далі Замовник і прийнято до виконання студентом Кушкою Михайлом Олександровичем, далі Виконавець, дата затвердження ТЗ.

* 1. Найменування і умовне позначення теми розробки

Найменування теми розробки – “Використання концепції ООП для створення гри на мові програмування Python”.

Умовне позначення теми розробки (шифр теми) – «ГРА 0.1».

1. Призначення розробки
   1. Функціональне призначення розробки

Функціональним призначенням програми є демонстрація візуалізації за допомогою мови програмування Python.

* 1. Експлуатаційне призначення

Програма може експлуатуватися в галузі комп’ютерних ігор, а також для розвитку дрібної моторики рухів.

1. Вимоги до програмного забезпечення
   1. Вимоги до функціональних характеристик
      1. Вимоги до складу виконуваних функцій

Програма повинна забезпечувати можливість виконання нижче наведених функцій:

1. можливість повертати космічний корабель на екрані
2. політ космічного корабля
3. випуск ракети космічним кораблем
4. знищення астероїдів на екрані
5. перехід на новий рівень
6. ввід імені гравця
7. збереження даних гравця до бази даних
8. вивід на екран трьох кращих гравців
   * 1. Вимоги до організації вхідних даних

Вхідними даними програми є ввід користувача, шляхом натискання на клавіатурі клавіші “пробіл” та стрілок. Після потрапляння астероїду в корабель гравця допускається також ввід імені, шляхом введення послідовності літер англійського алфавіту.

* + 1. Вимоги до організації вихідних даних

Вихідними даними є ім’я гравця та його рекорд. Ці дані зберігаються до бази даних для використання у подальших сесіях гри.

* 1. Вимоги до надійності
     1. Вимоги до надійного функціонування програми

Надійне функціонування програми повинно бути забезпечене виконанням замовником сукупності організаційно-технічних заходів, перелік яких наведено нижче:

1. організацією безперебійного струмопостачання технічних засобів;
2. використанням ліцензійного програмного забезпечення;
3. надати програмі доступ до директорії, де вона знаходиться для забезпечення дозапису даних в базу даних;
4. помістити усі додаткові файли, що постачаються разом з програмою в директорію до .exe файлу.
   * 1. Час відновлення після відмови

Час відновлення після відмови, що спричинена збоєм електропостачання технічних засобів, не крахом операційної системи, мусить не відрізнятися за часом від звичайного запуску програми, за умови виконання умов експлуатації технічних і програмних засобів.

Час відновлення після відмови, що спричинена неполадкою технічних засобів, крахом операційної системи, не повинно перевищувати часу, який потрібен на ліквідацію неполадок технічних засобів та переустановленням програмних засобів.

* + 1. Відмови через некоректні дії користувача

Відмови програми через некоректні дії користувача на етапі тестування програми виявлені не були.

* 1. Умови експлуатації
     1. Кліматичні умови експлуатації

Кліматичні умови експлуатації, при яких повинні забезпечуватися задані характеристики, повинні задовольняти вимогам, що заявлені до технічних засобів зокрема до умов їх експлуатації.

* + 1. Вимоги до видів обслуговування

Див. Вимоги до забезпечення надійного функціонування програми.

Програма не потребує будь-яких видів обслуговування.

* + 1. Вимоги до кількості та кваліфікації персоналу

Мінімальна кількість персоналу, що необхідний для роботи програми, повинна складати не менше 1 штатної одиниці – користувач системи з правами «Адміністратор».

Користувач програми з правами «Адміністратор» повинен мати навики роботи з .exe файлами та мінімальними навичками зміни прав файлів для надання прав читання, запису та виконання програмі та її додатковим файлам.

Персонал повинен бути атестованим на ІІ кваліфікаційну групу по електробезпеці (для роботи з кафедральним обладнанням).

* 1. Вимоги до складу і параметрів технічних засобів

До комп’ютера, на якому має бути розгорнуто програмне забезпечення, висуваються наступні вимоги:

1. 32 або 64-розрядний процесор з тактовою частотою не нижче 1.4 ГГц;
2. достатній об’єм оперативної пам’яті (не менше 1 ГБ);
3. достатній об’єм жорсткого диску (не менше 2 ГБ).
   1. Вимоги до інформаційної і програмної сумісності

Вимоги до додаткового ПЗ не висуваються.

* 1. Вимоги до маркування та упаковки

Вимоги до маркування та упаковки не висуваються.

* 1. Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

1. Вимоги до програмної документації
   1. Попередній склад програмної документації

Склад програмної документації повинен включати в себе:

1. технічне завдання;
2. керівництво користувача.
3. Техніко-економічні показники

Оскільки дана програма є комп’ютерною грою, то за умови правильного проведення реклами програмне забезпечення може приносити досить великий прибуток в порівнянні з затраченими на неї ресурсами.

1. Стадії та етапи розробки
   1. Стадії розробки

Розробка повинна бути проведена в три стадії:

1. розробка технічного завдання;
2. робоче проектування;
3. впровадження.
   1. Етапи розробки

На стадії розробки технічного завдання повинен бути виконаний етап розробки, узгодження і затвердження технічного завдання.

На стадії робочого проектування повинен бути виконаний наведений нижче перелік робіт:

1. розробка програми;
2. тестування програми;
3. дослідження програми;
4. розробка програмної документації.

На стадії впровадження виконується передача програми кінцевому замовнику.

* 1. Зміст роботи по етапам

На етапі розробки технічного завдання повинні бути виконані наступні роботи:

1. постановка задачі;
2. визначення і уточнення вимог до технічних засобів;
3. визначення вимог до програми;
4. визначення стадій, етапів і строків розробки програми та документації до неї;
5. вибір мови чи мов програмування;
6. узгодження та затвердження технічного завдання.

На етапі розробки програми повинні бути виконані роботи по програмуванню і налагодженню програми.

На етапі тестування повинні бути виконані роботи по тестуванню усіх основних синтаксичних конструкцій мови програмування, що інтерпретується, в обох режимах роботи програми.

На етапі досліджень програми повинні бути виконані наведені нижче види робіт:

1. розробка, узгодження і затвердження програми і методик дослідження;
2. проведення досліджень;
3. коригування програми і програмної документації за результатами досліджень.

На етапі розробки програмної документації повинна бути виконана розробка програмних документів.

На етапі підготовки та передачі програми повинна бути виконана робота по підготовці та передачі програми і програмної документації в експлуатацію на об’єктах кафедри АСОІУ.

1. Порядок контролю і прийому
   1. Види досліджень

Приймально-здавальні випробування програмного продукту мають проводитися згідно з розробленою виконавцем і узгодженою із замовником “Програмою та методикою випробувань” на об’єкті замовника.

* 1. Загальні вимоги до прийняття роботи

На основі проведених досліджень Виконавець спільно з Замовником підписують Акт прийому-здачі програми в експлуатацію.

Зміст

[Анотація 4](#_Toc502185109)

[1 Найменування та область застосування 8](#_Toc502185110)

[2 Підстави для розробки 9](#_Toc502185111)

[2.1 Перелік документів, на підставі яких ведеться розробка 9](#_Toc502185112)

[2.2 Найменування і умовне позначення теми розробки 9](#_Toc502185113)

[3 Призначення розробки 10](#_Toc502185114)

[3.1 Функціональне призначення розробки 10](#_Toc502185115)

[3.2 Експлуатаційне призначення 10](#_Toc502185116)

[4 Вимоги до програмного забезпечення 11](#_Toc502185117)

[4.1 Вимоги до функціональних характеристик 11](#_Toc502185118)

[4.1.1 Вимоги до складу виконуваних функцій 11](#_Toc502185119)

[4.1.2 Вимоги до організації вхідних даних 11](#_Toc502185120)

[4.1.3 Вимоги до організації вихідних даних 11](#_Toc502185121)

[4.2 Вимоги до надійності 11](#_Toc502185122)

[4.2.1 Вимоги до надійного функціонування програми 11](#_Toc502185123)

[4.2.2 Час відновлення після відмови 12](#_Toc502185124)

[4.2.3 Відмови через некоректні дії користувача 12](#_Toc502185125)

[4.3 Умови експлуатації 12](#_Toc502185126)

[4.3.1 Кліматичні умови експлуатації 12](#_Toc502185127)

[4.3.2 Вимоги до видів обслуговування 12](#_Toc502185128)

[4.3.3 Вимоги до кількості та кваліфікації персоналу 12](#_Toc502185129)

[4.4 Вимоги до складу і параметрів технічних засобів 13](#_Toc502185130)

[4.5 Вимоги до інформаційної і програмної сумісності 13](#_Toc502185131)

[4.6 Вимоги до маркування та упаковки 13](#_Toc502185132)

[4.7 Вимоги до транспортування та зберігання 13](#_Toc502185133)

[5 Вимоги до програмної документації 14](#_Toc502185134)

[5.1 Попередній склад програмної документації 14](#_Toc502185135)

[6 Техніко-економічні показники 15](#_Toc502185136)

[7 Стадії та етапи розробки 16](#_Toc502185137)

[7.1 Стадії розробки 16](#_Toc502185138)

[7.2 Етапи розробки 16](#_Toc502185139)

[7.3 Зміст роботи по етапам 16](#_Toc502185140)

[8 Порядок контролю і прийому 18](#_Toc502185141)

[8.1 Види досліджень 18](#_Toc502185142)

[8.2 Загальні вимоги до прийняття роботи 18](#_Toc502185143)

[Вступ 22](#_Toc502185144)

[9 Постановка задачі 23](#_Toc502185145)

[10 Теоретичні викладки 24](#_Toc502185146)

[11 Опис алгоритмів 25](#_Toc502185147)

[11.1 Загальний алгоритм 25](#_Toc502185148)

[11.2 Завершення гри 25](#_Toc502185149)

[12 Опис програмного забезпечення 26](#_Toc502185150)

[12.1 Опис діаграми класів програмного забезпечення 26](#_Toc502185151)

[12.2 Опис класів та їх методів 27](#_Toc502185152)

[12.2.1 Класи та їх методи 27](#_Toc502185153)

[12.2.2 Стандартні функції 31](#_Toc502185154)

[13 Тестування програмного забезпечення 33](#_Toc502185155)

[13.1 План тестування 33](#_Toc502185156)

[13.2 Приклади тестування 33](#_Toc502185157)

[14 Інструкція користувача 37](#_Toc502185158)

[14.1 Призначення програми 37](#_Toc502185159)

[14.2 Вимоги до системи 37](#_Toc502185160)

[14.3 Інструкція по роботі з програмою 37](#_Toc502185161)

[14.4 Склад програмного забезпечення 38](#_Toc502185162)

[Висновки 39](#_Toc502185163)

[Перелік посилань 40](#_Toc502185164)

[Додаток А 41](#_Toc502185165)

[screen.py 42](#_Toc502185166)

[interface.py 43](#_Toc502185167)

[game.py 50](#_Toc502185168)

[main.py 54](#_Toc502185169)

# Вступ

Завданням даної курсової роботи було створити гру з основними концепціями ООП на мові програмування Python [1], а саме: інкапсуляція, наслідування, поліморфізм.

Використання класів для розробки даного програмного забезпечення було дуже доречним, оскільки для розробки гри дуже зручно представляти окремі ігрові об’єкти як класи. Наприклад у даній програмі космічний корабель, астероїди та ракети – об’єкти окремих класів, що мають методи, унікальні для кожного класу. Також класами є й усі інші об’єкти такі, як рахунок гравця, написи на екрані та навіть сама гра є об’єктом класу Game.

# Постановка задачі

Розробити гру з використанням концепції ООП на мові програмування Python [1]. Для розробки ігрових компонентів був використаний модуль LiveWires [2], що базується на популярному модулі PyGame [3], який використовується для розробки ігор на мові програмування Python [1].

Гра складається з рівнів, котрі автоматично генеруються при проходженні попереднього, тобто їх є необмежена кількість. Основною метою гравця є знищити якомога більше астероїдів шляхом потрапляння ракетою по метеориту. При знищенні усіх астероїдів на поточному рівні гравець автоматично переходить на наступний.

У разі потраплянні астероїда в корабель відбувається знищення як астероїда, так і корабля та відбувається завершення гри. Власне завершенням гри є вивід відповідного повідомлення та прохання гравця ввести своє ім’я для збереження його результату до таблиці рекордів. Після чого відображається три найкращі результати з таблиці рекордів та поточний рахунок гравця.

Таблиця рекордів є текстовим файлом, у якому зберігаються такі дані, як ім’я гравця та рекорд, записані через пробіл. Дані кожного нового гравця записуються на новий рядок.

# Теоретичні викладки

В даній роботі були використані три основні концепції ООП, а саме: інкапсуляція, наслідування і поліморфізм.

За жанром дана гра є класичною, тобто дозволяє користувачеві згадати типові старі ігри такі, як Super Mario, Packman, Fling Tygers тощо. Це досягнуто за рахунок того, що гра написана без використання будь-яких програм для створення ігор, все в ній створювалося вручну. Наприклад вибух є не що інше, як кілька картинок з різними етапами вибуху, які відображаються досить швидко одна за одною, чим досягається анімація.

Кожен об’єкт програми представлений окремим класом. У програмі є наступні класи:

1. Wrapper, наслідується від класу Sprite модуля LiveWires [2]. Даний клас “огортає” екран і коректно опрацьовує вихід об’єктів за його межі.
2. Collider, наслідується від класу Wrapper. Обробляє такі події, як зіткнення двох об’єктів.
3. Asteroid, наслідується від класу Wrapper. Повністю відповідає за об’єкт астероїда на екрані.
4. Ship, наслідується від класу Collider. Відповідає за об’єкт корабля на екрані під час ігрового процесу.
5. Missile, наслідується від класу Collider. Відповідає за ракету, випущену з корабля.
6. Explosion, наслідується від класу Animation модуля LiveWires [2]. Даний клас робить анімацію вибуху об’єктів на екрані під час їх знищення.
7. Scores, наслідується від класу Text модуля LiveWires [2]. Відповідає за введення гравцем свого імені після завершення гри, збереження імені та рахунку до бази даних та відображення 3-х найкращих гравців на екрані.
8. Game. Власне відповідає за проведення самої гри. Тут створюються ігрові об’єкти та відбувається перехід на наступний рівень.

# Опис алгоритмів

## Загальний алгоритм

1. ПОЧАТОК
2. Задати розмір вікна програми і частоту оновлень екрану.
3. Зчитати і почати програвати музичний супровід.
4. Завантажити фон вікна програми.
5. Запуск основного циклу гри.
   1. Перехід на перший рівень.
      1. Створити метеорит та відобразити його на екрані.
      2. Відобразити напис з номером поточного рівня.
   2. ЯКЩО гравець потрапляє ракетою у астероїд.
      1. ЯКЩО розмір астероїду не найменший, ТО знищити астероїд І створити два менші астероїди, ІНАКШЕ знищити астероїд.
   3. ЯКЩО кількість астероїдів на екрані дорівнює нулю, ТО перейти на наступний рівень.
   4. ЯКЩО астероїд влучив у корабель, ТО завершити гру.
6. КІНЕЦЬ

## Завершення гри

1. ПОЧАТОК
2. Попросити гравця ввести своє ім’я.
3. ЯКЩО натиснуто <Enter>, ТО зчитати ім’я І зберегти ім’я і рахунок гравця до файлу результатів.
4. Вивести 3-х найкращих гравців на екран.
   1. Зчитати усі результати з файлу.
   2. Відсортувати усі результати за спаданням і вибрати з них перші 3.
   3. Відобразити на екрані імена і рахунки трьох найкращих гравців.
5. КІНЕЦЬ

# Опис програмного забезпечення

## Опис діаграми класів програмного забезпечення

Опис класів та їх зв’язків подано на рисунку 4.1 за допомогою UML діаграми класів.

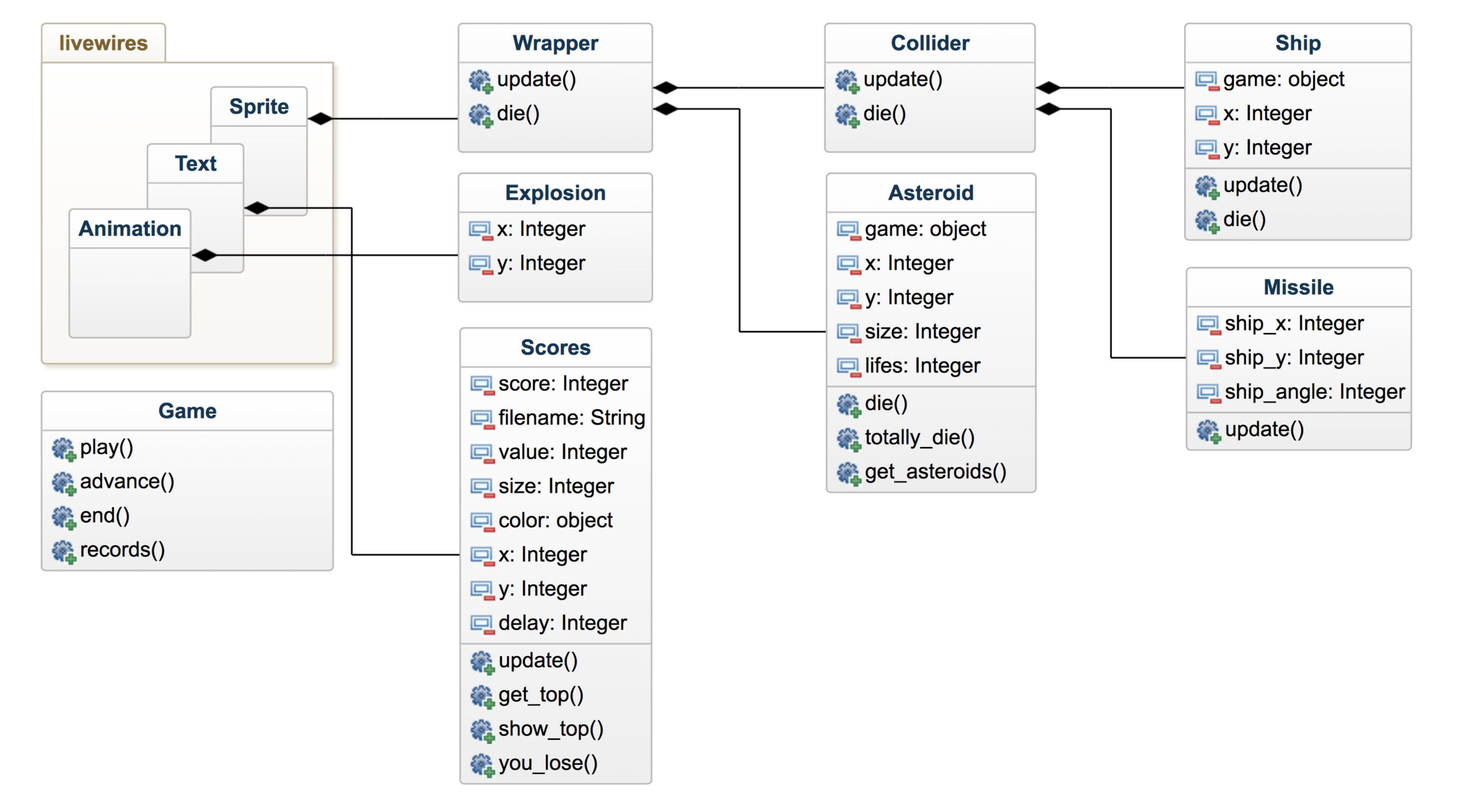


Рисунок 12.1 - UML діаграма класів

## Опис класів та їх методів

### Класи та їх методи

Опис медотів кожного класу наведений у таблиці 4.1

Таблиця 12.1 - Опис класів та їх методів

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 1 | Wrapper | update | Коректна обробка виходу об’єкту за межі екрану | - | - |
| 2 | Wrapper | die | Знищення об’єкту функції | - | - |
| 3 | Collider | update | Перевірка чи не зіштовхнулися об’єкти на екрані | - | - |
| 4 | Collider | die | Знищення об’єкта з анімацією | - | - |
| 5 | Explosion | \_\_init\_\_ | Анімація вибуху | Координати вибуху | - |
| 6 | Asteroid | \_\_init\_\_ | Ініціалізація основних параметрів класу | Об’єкт модуля games, координати астероїду, його розмір та кількість життів | - |
| 7 | Asteroid | die | У разі, якщо розмір астероїду найменший – самознищення, інакше – поділ на два менші астероїди | - | - |
| 8 | Asteroid | totally\_die | Знищення астероїду без додавання двох менших на екран | - | - |
| 9 | Asteroid | get\_asteroids | Повертає кількість астероїдів | - | К-сть астероїдів на екрані |
| 10 | Ship | \_\_init\_\_ | Ініціалізація основних параметрів класу | Об’єкт модуля games, координати корабля | - |
| 11 | Ship | update | Рух корабля в залежності від натиснутих клавіш, можливість випуску ракети | - | - |
| 12 | Ship | die | Знищення корабля, завершення гри | - | - |
| 13 | Missile | \_\_init\_\_ | Ініціалізація основних параметрів класу | Координати корабля, його кут повороту | - |
| 14 | Missile | update | Зменшення часу життя; зниження ракети, якщо час її існування закінчився | - | - |
| 15 | Scores | \_\_init\_\_ | Ініціалізація основних параметрів класу | Рахунок гравця, ім’я файлу рекордів, ім’я гравця, розмір та колір тексту, його положення на екрані та час існування | - |
| 16 | Scores | update | Введення імені користувача | - | - |
| 17 | Scores | get\_top | Визначення 3-х найкращих гравців | - | Список з 3-х найкращих гравців |
| 18 | Scores | show\_top | Відображення 3-х найкращих гравців | - | - |
| 19 | Scores | you\_lose | Збереження поточного рахунку і імені гравця до бази даних і відображення 3-х найкращих гравців | - | - |
| 20 | Game | \_\_init\_\_ | Ініціалізація основних параметрів класу | - | - |
| 21 | Game | play | Запускає ігровий процес | - | - |
| 22 | Game | advance | Перехід на наступний рівень | - | - |
| 23 | Game | end | Завершення гри | - | - |
| 24 | Game | records | Прохання гравця ввести своє ім’я та відображення 3-х найкращих гравців | - | - |

### Стандартні функції

Стандартні функції, що використані у проекті наведені у таблиці 4.2.

Таблиця 12.2 - Стандартні функції

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Модуль |
| 1 | destroy | Знищення об’єкта класу | games |
| 2 | super | Виклик методу базового класу | <за замовчуванням> |
| 3 | screen.add | Додати об’єкт на екран | games |
| 4 | load\_sound | Завантажити звук з файлу у програму | games |
| 5 | sound.play | Програвання звуку | games |
| 6 | choice([<lst>]) | Вибір довільного значення зі списку | random |
| 7 | random | Вибір довільного дійсного числа від -1 до 1 | random |
| 8 | int | Зведення числа до цілого формату | <за замовчуванням> |
| 9 | range | Діапазон значень | <за замовчуванням> |
| 10 | append | Додавання значення до списку | <за замовчуванням> |
| 11 | load\_image | Завантажити зображення з файлу у програму | games |
| 12 | keyboard.is\_pressed | Перевіряє чи натиснута клавіша | games |
| 13 | pi, sin, cos | Математичні функції | math |
| 14 | min, max | Мінімальне/максимальне значення зі списку | <за замовчуванням> |
| 15 | str, chr | Приведення до рядкового, символьного типу | <за замовчуванням> |
| 16 | len | Визначення довжини переданого рядка, списку тощо | <за замовчуванням> |
| 17 | split | Розділити рядок по розділювачу | <за замовчуванням> |
| 18 | sorted | Повернення відсортованого списку (не змінює вхідний список) | <за замовчуванням> |
| 19 | itemgetter(<i>) | Сортування за i-ю розмірністю | operator |
| 20 | open | Відкрити файл | <за замовчуванням> |
| 21 | write | Запис у файл | <за замовчуванням> |
| 22 | randrange | Вибір довільного значення з проміжку | random |

# Тестування програмного забезпечення

## План тестування

Тестування програми відбувається після її розробки для виявлення помилок в логіці програми та її реалізації. Наведений нижче план тестування перевіряє усі “слабкі місця” програми.

1. Перевірка можливості повороту корабля ліворуч та праворуч.
2. Перевірка можливості польоту корабля.
3. Перевірка можливості випускання ракет кораблем.
4. Спроба затиснути пробіл з метою отримання неперервного потоку ракет.
5. Спроба розігнати корабель до нескінченної швидкості.
6. Спроба введення дуже довгого імені гравця.
7. Спроба залишити введене ім’я порожнім.

## Приклади тестування

Тестування програмного забезпечення було проведено згідно з планом, наведеним у підрозділі 5.1. Були отримані наступні результати.

1. При натисканні на клавіатурі стрілок “праворуч” та “ліворуч” корабель повертався у відповідні сторони.
2. При натисканні стрілки “вгору” корабель починає розганятися та летіти. А відповідно при натисканні стрілки “вниз” – корабель гальмує.
3. Якщо користувач натискає клавішу “пробіл”, то корабель випускає ракету. Скріншот можна побатити на рисунку 5.1.



Рисунок 13.1 - Випускання ракети кораблем

1. Спроба затиснути пробіл з метою отримання неперервного потоку ракет.

Корабель може випускати ракету не частіше, ніж кожні 0.4 секунди.

1. Спроба розігнати корабель до нескінченної швидкості.

Максимальна швидкість корабля 200 пікселів на секунду. Дане обмеження швидкості дозволяє уникнути втрати контролю кораблем.

1. Спроба введення дуже довгого імені гравця.

Обмеження довжини імені складає 8 символів, а 8 символів при даному розмірі тексту завжди поміститься на екран. Скріншот цього обмеження можна побачити на рисунку 5.2.



Рисунок 13.2 - Спроба введення надто довгого імені

1. Залишити введене ім’я порожнім.

При спробі залишити поле для вводу імені порожнім гравцю присвоюється ім’я player. Скріншот даного “запобіжника” можна побачити на рисунку 5.3.



Рисунок 13.3 - Запобігання невводу імені

# Інструкція користувача

## Призначення програми

Дана програма є грою, тому її основним призначенням є розвага користувача. Як другорядне призначення в контексті її програмної реалізації можна виділити здобуття навичок у використанні ООП за допомогою мови програмування Python.

## Вимоги до системи

1. Intel® Core 2 або AMD Athlon® 64 процесор; частота процесора 1.5 GHz або швидше
2. Microsoft Windows XP чи більш пізня версія
3. 1 GB або більше RAM (2 GB рекомендовано)
4. 2 GB або більше вільного місця на диску
5. 640 x 480 дисплей (1280x800 рекомендовано)

## Інструкція по роботі з програмою

Інтерфейс програми наведений на рисунку 6.1.

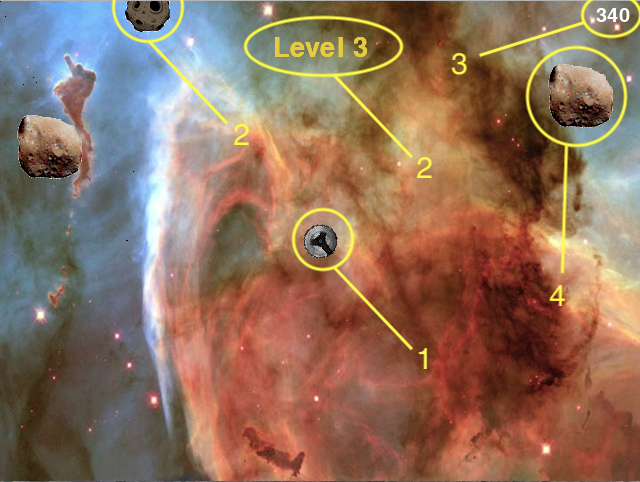


Рисунок 14.1 - Інтерфейс програми

1 – космічний корабель

2 – номер рівня (показується лише при переході на наступний рівень)

3 – рахунок гравця

4 – метеорит

5 – міцний метеорит (потрібно 3 потрапляння замість 1 для того, щоб його зруйнувати)

Також у програмі використовуються наступні клавіші:

1. Пробіл – випуск ракети з “носу” корабля.
2. Стрілка вгору – збільшити швидкість корабля.
3. Стрілка вниз – зменшити швидкість корабля.
4. Стрілка ліворуч/праворуч – повернути корабель ліворуч/праворуч.

## Склад програмного забезпечення

Програмне забезпечення складається з одного файлу .exe, який власне і є програмою. Також разом з .exe файлом у тій самій директорії мусить бути папка database, яка використовується для збереження результатів гравців. Окрім того у даній директорії мусять бути й інші файли, які постачаються розробником разом з .exe файлом та містять у собі різноманітні модулі та скрипти, необхідні для роботи програми.

Завантажити гру та знайти детальну інформацію до неї можна на моєму GitHub [4].

# Висновки

На аналітичному етапі розробки програмного забезпечення були детально розглянуті та вивчені основні концепції ООП та їх реалізація на мові програмування Python.

Згодом, при проектуванні програмного забезпечення, було вирішено які ігрові компоненти кожен клас реалізовуватиме, від кого який клас буде наслідуватися та способи взаємодії між ними. Пізніше було вирішено створити окремий клас Game, котрий міститиме сам основний ігровий процес для максимального спрощення основної програми.

Після завершення написання програмного забезпечення був проведений детальний аналіз програми та її модулів, здійснений невеликий рефакторинг коду для більш елегантної реалізації деяких підзадач. Окрім того було проведене детальне тестування гри на наявність у ній помилок, що можуть призвести до аварійного завершення роботи програми, чи просто помилок у логіці роботи програмного забезпечення. Після тестування знайдені недоліки було усунуто.

При подальшому тривалому тестуванню програми додаткових помилок виявлено не було, що зводить до мінімуму виявлення помилок у роботі програмного забезпечення у майбутньому. Також не було помічено будь-якої незручної чи неочевидної поведінки програми у відповідь на дії користувача.

# Перелік посилань

1. Документація з Python – https://docs.python.org/2/
2. Модуль LiveWires на GitHub – https://github.com/livewires/python/tree/master/module
3. Документація з модулю PyGame – https://www.pygame.org/docs/
4. Гра на GitHub - https://github.com/kushkamisha/Space-game

# Додаток А

(Найменування програми (документа))

*Тексти програмного коду програмного забезпечення вирішення задачі використання концепції ООП для створення гри*

*на мові на мові програмування Python*

(Вид носія даних)

*CD-RW*

(Обсяг програми (документа), арк., Кб)

*13 арк, 40 Кб*

*студента групи ІП-61 ІІ курсу*

Кушки М.О.

screen.py

from livewires import games

games.init(screen\_width=640, screen\_height=480, fps=50)

class Wrapper(games.Sprite):

""" Sprite, which wraps around the screen """

def update(self):

""" Wrap sprite around screen """

if self.top > games.screen.height:

self.bottom = 0

if self.bottom < 0:

self.top = games.screen.height

if self.left > games.screen.width:

self.right = 0

if self.right < 0:

self.left = games.screen.width

def die(self):

""" Destroys object """

self.destroy()

class Collider(Wrapper):

""" A wraper that can collide with another object """

def update(self):

""" Check for overlapping sprites """

super(Collider, self).update()

if self.overlapping\_sprites:

for sprite in self.overlapping\_sprites:

sprite.lifes -= 1

if sprite.lifes == 0:

sprite.die()

self.die()

def die(self):

""" Destroys object with an explosion """

new\_explosion = Explosion(x=self.x, y=self.y)

games.screen.add(new\_explosion)

self.destroy()

class Explosion(games.Animation):

""" Animationed explosion """

sound = games.load\_sound("sounds/explosion.wav")

images = ["images/explosion1.bmp",

"images/explosion2.bmp",

"images/explosion3.bmp",

"images/explosion4.bmp",

"images/explosion5.bmp",

"images/explosion6.bmp",

"images/explosion7.bmp",

"images/explosion8.bmp",

"images/explosion9.bmp"]

def \_\_init\_\_(self, x, y):

super(Explosion, self).\_\_init\_\_(images=Explosion.images,

x=x, y=y,

repeat\_interval=2, n\_repeats=1,

is\_collideable=False)

Explosion.sound.play()

interface.py

from livewires import games, color

import random

import math

import operator

from screen import Wrapper, Collider

class Asteroid(Wrapper):

""" Moving asteroid on the screen """

SMALL = 1

MEDIUM = 2

LARGE = 3

POWERFUL = 1.5

SPAWN = 2

POINTS = 30

total = 0

images = {SMALL: games.load\_image("images/asteroid\_small.bmp"),

MEDIUM: games.load\_image("images/asteroid\_med.bmp"),

LARGE: games.load\_image("images/asteroid\_big.bmp"),

POWERFUL: games.load\_image("images/asteroid\_powerful.bmp")}

SPEED = 2

def \_\_init\_\_(self, game, x, y, size, lifes):

""" Initialize sprite with asteroid image """

self.lifes = lifes

Asteroid.total += 1

self.asteroids = []

super(Asteroid, self).\_\_init\_\_(

image=Asteroid.images[size],

x=x, y=y,

dx=random.choice([-1, 1]) \* Asteroid.SPEED \*

random.random() / size \* 2.5,

dy=random.choice([-1, 1]) \* Asteroid.SPEED \*

random.random() / size \* 2.5)

self.game = game

self.size = size

def die(self):

""" Destroys asteroid """

Asteroid.total -= 1

self.game.score.value += int(Asteroid.POINTS / self.size)

self.game.score.right = games.screen.width - 10

# if size of asteroid isn't small, replace with two smaller asteroids

if self.size != Asteroid.SMALL and self.size != Asteroid.POWERFUL:

for i in range(Asteroid.SPAWN):

new\_asteroid = Asteroid(game=self.game,

x=self.x,

y=self.y,

size=self.size - 1,

lifes=1)

games.screen.add(new\_asteroid)

self.asteroids.append(new\_asteroid)

# if all asteroids are gone, advance to next level

if Asteroid.total == 0:

self.game.advance()

super(Asteroid, self).die()

def totally\_die(self):

"""

Destroys asteroid without adding smaller asteroids on the screen

"""

super(Asteroid, self).die()

def get\_asteroids(self):

""" Returns list of asteroids on the screen """

return self.asteroids

class Ship(Collider):

""" Player's ship """

image = games.load\_image("images/ship.bmp")

sound = games.load\_sound("sounds/thrust.wav")

ROTATION\_STEP = 5

VELOCITY\_STEP = .07

MISSILE\_DELAY = 20

VELOCITY\_MAX = 4

def \_\_init\_\_(self, game, x, y):

""" Initialize space ship sprite """

super(Ship, self).\_\_init\_\_(image=Ship.image, x=x, y=y)

self.game = game

self.missile\_wait = 0

def update(self):

""" Rotate based on key pressed """

super(Ship, self).update()

# rotate based on left and right arrow keys

if games.keyboard.is\_pressed(games.K\_LEFT):

self.angle -= Ship.ROTATION\_STEP

if games.keyboard.is\_pressed(games.K\_RIGHT):

self.angle += Ship.ROTATION\_STEP

# move ship

if games.keyboard.is\_pressed(games.K\_UP):

Ship. sound.play()

# change velocity components based on ship's angle

angle = self.angle \* math.pi / 180 # convert to radians

self.dx += Ship.VELOCITY\_STEP \* math.sin(angle)

self.dy += Ship.VELOCITY\_STEP \* -math.cos(angle)

# cap velocity in each direction

self.dx = min(max(self.dx, -Ship.VELOCITY\_MAX), Ship.VELOCITY\_MAX)

self.dy = min(max(self.dy, -Ship.VELOCITY\_MAX), Ship.VELOCITY\_MAX)

if games.keyboard.is\_pressed(games.K\_DOWN):

Ship. sound.play()

# change velocity components based on ship's angle

angle = self.angle \* math.pi / 180 # convert to radians

self.dx -= Ship.VELOCITY\_STEP \* math.sin(angle)

self.dy -= Ship.VELOCITY\_STEP \* -math.cos(angle)

# cap velocity in each direction

self.dx = min(max(self.dx, -Ship.VELOCITY\_MAX), Ship.VELOCITY\_MAX)

self.dy = min(max(self.dy, -Ship.VELOCITY\_MAX), Ship.VELOCITY\_MAX)

# if waiting until the ship can fire next, decrease wait

if self.missile\_wait > 0:

self.missile\_wait -= 1

# if pressed space and missile wait is over, then launch missile

if games.keyboard.is\_pressed(games.K\_SPACE) and self.missile\_wait == 0:

new\_missile = Missile(self.x, self.y, self.angle)

games.screen.add(new\_missile)

self.missile\_wait = Ship.MISSILE\_DELAY

def die(self):

""" Destroy ship and end the game """

self.game.end()

super(Ship, self).die()

class Missile(Collider):

""" A missile launched by the player's ship """

image = games.load\_image("images/missile.bmp")

sound = games.load\_sound("sounds/missile.wav")

BUFFER = 40

VELOCITY\_FACTOR = 12

LIFETIME = 30

def \_\_init\_\_(self, ship\_x, ship\_y, ship\_angle):

""" Initialize sprite with missile image """

Missile.sound.play()

# convert to radians

angle = ship\_angle \* math.pi / 180

# calculate missile's starting position

buffer\_x = Missile.BUFFER \* math.sin(angle)

buffer\_y = Missile.BUFFER \* -math.cos(angle)

x = ship\_x + buffer\_x

y = ship\_y + buffer\_y

# calculate missile's velocity components

dx = Missile.VELOCITY\_FACTOR \* math.sin(angle)

dy = Missile.VELOCITY\_FACTOR \* -math.cos(angle)

# create the missile

super(Missile, self).\_\_init\_\_(image=Missile.image,

x=x, y=y,

dx=dx, dy=dy)

self.lifetime = Missile.LIFETIME

def update(self):

""" Move the missile """

super(Missile, self).update()

# if lifetime is up, destroy the missile

self.lifetime -= 1

if self.lifetime == 0:

self.destroy()

class Scores(games.Text):

""" Save/display top scores after end of the game. """

def \_\_init\_\_(self, score,

filename,

value, size=60,

color=color.black,

x=games.screen.width/2,

y=games.screen.height/2,

delay=5):

super(Scores, self).\_\_init\_\_(value=value,

size=size,

color=color,

x=x, y=y)

self.default\_length = len(self.value)

self.filename = filename

self.score = str(score)

self.delay = delay

self.time\_remain = delay

self.top\_players = []

def update(self):

""" Ley user enter his nickname. """

if (self.time\_remain != 0):

self.time\_remain -= 1

alphabet = [ch\_code for ch\_code in range(97, 122)]

numbers = [ch\_code for ch\_code in range(48, 58)]

# Enter letters

for char\_code in alphabet:

if games.keyboard.is\_pressed(char\_code) and \

self.time\_remain == 0 and len(self.value) < 25:

self.value += chr(char\_code).upper()

self.time\_remain = self.delay

# Enter digits

for char\_code in numbers:

if games.keyboard.is\_pressed(char\_code) and \

self.time\_remain == 0 and len(self.value) < 25:

self.value += chr(char\_code).upper()

self.time\_remain = self.delay

# Enter backspace

if games.keyboard.is\_pressed(games.K\_BACKSPACE) and \

self.time\_remain == 0 and len(self.value) > 17:

self.value = self.value[:-1]

self.time\_remain = self.delay

# Enter return

if games.keyboard.is\_pressed(games.K\_RETURN):

# Save score to database and display top scores

# If username is empty set is to PLAYER

if (len(self.value) == self.default\_length):

self.value += 'PLAYER'

self.you\_lose()

def get\_top(self):

""" Get top 3 players from file """

scores\_arr = []

with open(self.filename) as scores:

for player in scores:

player = player[:-1]

temp = [player.split()[0], int(player.split()[1])]

scores\_arr.append(temp)

top3players\_int = sorted(scores\_arr,

key=operator.itemgetter(1),

reverse=True)[:3]

top3players = []

for i in range(len(top3players\_int)):

top3players.append([top3players\_int[i][0],

str(top3players\_int[i][1])])

return top3players

def show\_top(self):

""" Display top scores """

self.top\_players = self.get\_top()

for i in range(len(self.top\_players)):

# Add "Top players" phrase on the screen

phrase = games.Text(value="TOP PLAYERS",

size=self.size,

color=color.red,

x=games.screen.width/2,

y=70)

games.screen.add(phrase)

# Add player's name on the screen

player = games.Text(value=self.top\_players[i][0],

size=self.size,

color=self.color,

x=250,

y=150+50\*i)

games.screen.add(player)

# Add score on the screen

score = games.Text(value=self.top\_players[i][1],

size=self.size,

color=self.color,

x=450,

y=150+50\*i)

games.screen.add(score)

# Add your name on the screen

label = games.Text(value="Your score: ",

size=self.size,

color=self.color,

x=250,

y=330)

games.screen.add(label)

# Add your score on the screen

my\_points = games.Text(value=self.score,

size=self.size,

color=self.color,

x=450,

y=330)

games.screen.add(my\_points)

# Add <press Esc to exit> label

label = games.Text(value="<press Esc to exit>",

size=40,

color=self.color,

x=games.screen.width/2,

y=410)

games.screen.add(label)

def you\_lose(self):

""" Save score to database and display top scores """

with open(self.filename, 'a') as f:

# Save score to the file

f.write(self.value[17:] + ' ' + self.score + '\n')

self.destroy()

self.show\_top()

game.py

import random

from livewires import games, color

from interface import Asteroid, Ship, Scores

class Game():

""" The game """

def \_\_init\_\_(self):

""" Initialize game object """

# set level

self.level = 0

# cheate list of asteroids

self.asteroids = []

# load sound for level advance

self.sound = games.load\_sound("sounds/level.wav")

# create score

self.score = games.Text(value=0,

size=30,

color=color.white,

top=5,

right=games.screen.width-19,

is\_collideable=False)

games.screen.add(self.score)

# create player's ship

self.ship = Ship(game=self,

x=games.screen.width/2,

y=games.screen.height/2)

games.screen.add(self.ship)

def play(self):

""" Starts game """

# begin music theme

games.music.load("sounds/theme.mid")

games.music.play(-1)

# load and set background

nebula\_image = games.load\_image("images/nebula.png")

games.screen.background = nebula\_image

# advance to level 1

self.advance()

# start game

games.screen.mainloop()

def advance(self):

""" Advance to the next level """

self.level += 1

# reserved space around the ship

BUFFER = 150

# create new asteroids

for i in range(self.level):

# calculate an x and y at least BUFFER distance from the ship

# choose minimum distance along x-axis and y-axis

x\_min = random.randrange(BUFFER)

y\_min = BUFFER - x\_min

# choose distance along x-axis and y-axis based on minimum distance

x\_distance = random.randrange(x\_min, games.screen.width - x\_min)

y\_distance = random.randrange(y\_min, games.screen.height - y\_min)

# calculate location based on distance

x = self.ship.x + x\_distance

y = self.ship.y + y\_distance

# wrap around screen, if necessary

x %= games.screen.width

y %= games.screen.height

# create the asteroid

if i % 2 == 0:

size = Asteroid.LARGE

lifes = 1

else:

size = Asteroid.POWERFUL

lifes = 3

new\_asteroid = Asteroid(game=self,

x=x, y=y,

size=size,

lifes=lifes)

games.screen.add(new\_asteroid)

self.asteroids.append(new\_asteroid)

# display level number

level\_message = games.Message(value="Level "+str(self.level),

size=40,

color=color.yellow,

x=games.screen.width/2,

y=games.screen.height/10,

lifetime=3\*games.screen.fps,

is\_collideable=False)

games.screen.add(level\_message)

# play new level sound (except at first level)

if self.level > 1:

self.sound.play()

def end(self):

""" Ends game """

# Destroy all asteroids on the screen

for asteroid in self.asteroids:

children = asteroid.get\_asteroids()

for child in children:

child.totally\_die()

asteroid.totally\_die()

# show 'Game over' for 1 second

end\_message = games.Message(value="Game over",

size=90,

color=color.red,

x=games.screen.width/2,

y=games.screen.height/2,

lifetime=1\*games.screen.fps,

after\_death=self.records,

is\_collideable=False)

games.screen.add(end\_message)

def records(self):

""" Enter player's name and display top 3 players """

scores = Scores(score=str(self.score.value),

filename="database/scores.txt",

value="Enter your name: ")

games.screen.add(scores)

main.py

# # # # # # # # # #

# Astrocrash game #

# # # # # # # # # #

from game import Game

def main():

astrocrash = Game()

astrocrash.play()

if \_\_name\_\_ == "\_\_main\_\_":

main()