**Анотація**

У курсовій роботі розглянуто основні аспекти створення гри мовою **Python** використовуючи бібліотеку **pygame**.

Зміст

Розділ 1. Основні технології

**1.1 Мова програмування Python5**

1.1.1 Загальна характеристика5

1.1.2 Філософія5

1.1.3 Історія6

**1.2 Бібліотека pygame8**

1.2.1 Загальна характеристика8

1.2.2 Історія8

**1.3 Допоміжні бібліотеки8**

1.3.1 Pickle8

1.3.2 Pyinstaller9

Розділ 2. Опис створеного програмного продукту

**2.1 Постановка задачі10**

**2.2 Структура10**

2.2.1 Структура проекту11

2.2.2 Файлова структура12

**2.3 Базові модулі12**

1.2.1 Sprite12

1.2.2 Surface14

**2.4 Модулі16**

2.4.1 Command16

2.4.2 Controls18

2.4.3 Program19

2.4.4 Robot20

2.4.5 Tile20

2.4.6 Scene21

2.4.7 Menu 21

2.4.8 Message22

**2.5 Додаткові файли23**

2.4.1 Load23

2.4.2 Level24

2.4.3 Creator24

2.5.3 Variables24

**2.5 Основний двигун25**

**Вступ**

Сьогодні для дітей комп'ютер є невід’ємною частиною до­звіл­ля. Деяким він до вподоби на стільки, що вже змалечку вони пла­нують пов'язати з ним своє життя. Тому варто в першу чергу донести до юного розуму, що таке комп'ютер, і як з ним спілкуватися.

В цьому полягає ідея роботи: створити цікаву гру, яка б поєднувала в собі програмування та розвагу.

Завдання користувача: провести робота з точки **А** в точку **В** за допомогою обмеженої кількості команд. В міру наявності тих, чи інших команд, складність гри може варіюватися від дитячої, до дуже складної.

**Розділ 1. Основні технології**

* 1. ***Мова програмування Python***
     1. ***Загальна характеристика.***

**Python** — [інтерпретована](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) [об'єктно-орієнтована](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) [мова програмування](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) високого рівня з [строгою](https://uk.wikipedia.org/w/index.php?title=%D0%A1%D1%82%D1%80%D0%BE%D0%B3%D0%B0_%D1%82%D0%B8%D0%BF%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F&action=edit&redlink=1) [динамічною типізацією](https://uk.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D1%96%D1%87%D0%BD%D0%B0_%D1%82%D0%B8%D0%BF%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F). Розроблена в [1990](https://uk.wikipedia.org/wiki/1990) році[*Гвідо ван Россумом*](https://uk.wikipedia.org/wiki/%D0%93%D0%B2%D1%96%D0%B4%D0%BE_%D0%B2%D0%B0%D0%BD_%D0%A0%D0%BE%D1%81%D1%81%D1%83%D0%BC). [Структури даних](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85) високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання існуючих компонентів. **Python** підтримує [модулі](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор **Python** та стандартні бібліотеки доступні як у скомпільованій так і у вихідній формі на всіх основних платформах. В мові програмування **Python** підтримується декілька [парадигм програмування](https://uk.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), зокрема: [*об'єктно-орієнтована*](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)*,*[*процедурна*](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)*,*[*функціональна*](https://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)*та*[*аспектно-орієнтована*](https://uk.wikipedia.org/wiki/%D0%90%D1%81%D0%BF%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F).

***1.1.2 Переваги мови***

Серед основних її переваг можна виділити такі:

* чистий [синтаксис](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81) (для виділення блоків слід використовувати відступи);
* переносність програм (що властиве більшості інтерпретованих мов);
* стандартний [дистрибутив](https://uk.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D0%B8%D0%B2) має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу);
* можливість використання **Python** в діалоговому режимі (дуже корисне для експериментування та розв'язання простих задач);
* стандартний дистрибутив має просте, але разом із тим досить потужне [середовище розробки](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D1%81%D0%B5%D1%80%D0%B5%D0%B4%D0%BE%D0%B2%D0%B8%D1%89%D0%B5_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B8), яке зветься **IDLE** і яке написане на мові **Python**;
* зручний для розв'язання математичних проблем (має засоби роботи з [комплексними числами](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BB%D0%B5%D0%BA%D1%81%D0%BD%D1%96_%D1%87%D0%B8%D1%81%D0%BB%D0%B0), може оперувати з цілими числами довільної величини, у діалоговому режимі може використовуватися як потужний [калькулятор](https://uk.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BB%D1%8C%D0%BA%D1%83%D0%BB%D1%8F%D1%82%D0%BE%D1%80));
* відкритий код (можливість редагувати його іншими користувачами).

**Python** має ефективні [структури даних](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85) високого рівня та простий, але ефективний підхід до [об'єктно-орієнтованого програмування](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F). Елегантний синтаксис **Python**, динамічна обробка типів, а також те, що це [інтерпретована мова](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0), роблять її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ.

Інтерпретатор мови **Python** і багата [стандартна бібліотека](https://uk.wikipedia.org/w/index.php?title=%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D0%B0_%D0%B1%D1%96%D0%B1%D0%BB%D1%96%D0%BE%D1%82%D0%B5%D0%BA%D0%B0&action=edit&redlink=1) (як вихідні тексти, так і бінарні дистрибутиви для всіх основних [операційних систем](https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0)) можуть бути отримані з сайту **Python** [*www.python.org*](http://www.python.org/), і можуть вільно розповсюджуватися. Цей самий сайт має дистрибутиви та посилання на численні модулі, програми, утиліти та додаткову документацію.

Інтерпретатор мови **Python** може бути розширений функціями та типами даних, розробленими на [**C**](https://uk.wikipedia.org/wiki/C_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) чи [**C＋＋**](https://uk.wikipedia.org/wiki/%D0%A1%D1%96-%D0%BF%D0%BB%D1%8E%D1%81-%D0%BF%D0%BB%D1%8E%D1%81) (або на іншій мові, яку можна викликати із **C**). **Python** також зручна як мова розширення для прикладних програм, що потребують подальшого налагодження.

**Python** [портований](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) і працює майже на всіх відомих платформах — від [КПК](https://uk.wikipedia.org/wiki/%D0%9A%D0%B8%D1%88%D0%B5%D0%BD%D1%8C%D0%BA%D0%BE%D0%B2%D0%B8%D0%B9_%D0%BF%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80) до [мейнфреймів](https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B9%D0%BD%D1%84%D1%80%D0%B5%D0%B9%D0%BC). Існують порти під [Microsoft Windows](https://uk.wikipedia.org/wiki/Microsoft_Windows), всі варіанти [UNIX](https://uk.wikipedia.org/wiki/UNIX) (включаючи [FreeBSD](https://uk.wikipedia.org/wiki/FreeBSD) та [GNU/Linux](https://uk.wikipedia.org/wiki/GNU/Linux)), [Plan 9](https://uk.wikipedia.org/wiki/Plan_9), [Mac OS](https://uk.wikipedia.org/wiki/Mac_OS) та [Mac OS X](https://uk.wikipedia.org/wiki/Mac_OS_X), [iPhone OS](https://uk.wikipedia.org/wiki/IPhone_OS) 2.0 і вище, [Palm OS](https://uk.wikipedia.org/wiki/Palm_OS), [OS/2](https://uk.wikipedia.org/wiki/OS/2), [Amiga](https://uk.wikipedia.org/wiki/Amiga), [AS/400](https://uk.wikipedia.org/wiki/IBM_System_i) та навіть [OS/390](https://uk.wikipedia.org/w/index.php?title=OS/390&action=edit&redlink=1)[[en]](https://en.wikipedia.org/wiki/OS/390), [Symbian](https://uk.wikipedia.org/wiki/Symbian) та [Android](https://uk.wikipedia.org/wiki/Android)[[13]](https://uk.wikipedia.org/wiki/Python#cite_note-13).

При цьому, на відміну від багатьох портованих систем, для всіх основних платформ **Python** має підтримку характерних для даної платформи технологій (наприклад, Microsoft [COM](https://uk.wikipedia.org/wiki/Microsoft_Component_Object_Model)/[DCOM](https://uk.wikipedia.org/wiki/Microsoft_Distributed_Component_Object_Model)). Навіть більше, існує спеціальна версія **Python** для [**віртуальної машини Java**](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_Java) — [**Jython**](https://uk.wikipedia.org/wiki/Jython), що дозволяє інтерпретатору виконуватися на будь-якій системі, яка підтримує **Java**, при цьому класи **Java** можуть безпосередньо використовуватися з **Python** й навіть бути написаними на ньому. Також кілька проектів забезпечують інтеграцію з платформою [**Microsoft.NET**](https://uk.wikipedia.org/wiki/Microsoft.NET), основні з яких — [**IronPython**](https://uk.wikipedia.org/wiki/IronPython) та [**Python.Net**](https://uk.wikipedia.org/w/index.php?title=Python.Net&action=edit&redlink=1).

* + 1. ***Філософія***

Розробники мови **Python** є прихильниками певної філософії програмування, яку називають *«The Zen of Python» (*[*«Дзен*](https://uk.wikipedia.org/wiki/%D0%94%D0%B7%D0%B5%D0%BD)*Пайтона»).* Її текст можна отримати у [інтерпретаторі](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80) **Python** за допомогою команди *import this* (лише один раз за сесію). Автором цієї філософії вважається [*Тім Пейтерс*](https://uk.wikipedia.org/w/index.php?title=%D0%A2%D1%96%D0%BC_%D0%9F%D0%B5%D0%B9%D1%82%D0%B5%D1%80%D1%81&action=edit&redlink=1).

Текст філософії:

* Гарне краще за потворне.
* Очевидне краще за неочевидне.
* Просте краще за складне.
* Складне краще за заплутане.
* Плоске краще за вкладене.
* Розділене краще за суцільне.
* Легкість читання має значення.
* Особливі випадки не є настільки особливими, щоб порушувати правила.
* Хоча практичність є важливішою за бездоганність.
* Помилки ніколи не повинні бути замовчуваними.
* Хіба що замовчуваними відверто.
* Зустрівши двозначність, відкиньмо спокусу вгадати.
* Має бути один — і, бажано, *тільки* один — очевидний спосіб зробити це.
* Хоча спочатку він може бути не очевидним, якщо ви не голландець.
* Зараз — краще, ніж ніколи.
* Хоча ніколи, найчастіше, — краще, ніж *просто* зараз.
* Якщо реалізацію важко пояснити — задум поганий.
* Якщо реалізацію легко пояснити — *можливо*, задум добрий.
* Простори імен — чудова річ, тож робімо їх більше!
  + 1. ***Історія***

Розробка мови **Python** була розпочата в кінці [1980-х років](https://uk.wikipedia.org/wiki/1980-%D1%82%D1%96)[[5]](https://uk.wikipedia.org/wiki/Python#cite_note-5) співробітником голландського інституту **CWI** [*Гвідо ван Россумом*](https://uk.wikipedia.org/wiki/%D0%93%D0%B2%D1%96%D0%B4%D0%BE_%D0%B2%D0%B0%D0%BD_%D0%A0%D0%BE%D1%81%D1%81%D1%83%D0%BC). Для розподіленої *ОС*[*Amoeba*](https://uk.wikipedia.org/wiki/Amoeba_(%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0)) потрібна була розширювана скриптова мова, і *Гвідо* почав писати **Python** на дозвіллі, запозичивши деякі напрацювання для мови [**ABC**](https://uk.wikipedia.org/w/index.php?title=ABC_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)&action=edit&redlink=1) (*Гвідо* брав участь у розробці цієї мови, орієнтованої на навчання програмування). У лютому [1991](https://uk.wikipedia.org/wiki/1991) року *Гвідо* опублікував вихідний текст в [групі новин](https://uk.wikipedia.org/wiki/%D0%93%D1%80%D1%83%D0%BF%D0%B0_%D0%BD%D0%BE%D0%B2%D0%B8%D0%BD) **alt.sources**. Мова почала вільно поширюватися через [Інтернет](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82), і сподобалася іншим [програмістам](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%96%D1%81%D1%82). З [1991](https://uk.wikipedia.org/wiki/1991) року **Python** є цілком [об'єктно-орієнтованим](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F). **Python** також запозичив багато рис таких мов, як [**C**](https://uk.wikipedia.org/wiki/C_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), [**C++**](https://uk.wikipedia.org/wiki/C%2B%2B), [**Modula-3**](https://uk.wikipedia.org/w/index.php?title=Modula-3&action=edit&redlink=1) і [**Icon**](https://uk.wikipedia.org/w/index.php?title=Icon_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)&action=edit&redlink=1), й окремі риси [функціонального програмування](https://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) з [Ліспу](https://uk.wikipedia.org/wiki/%D0%9B%D1%96%D1%81%D0%BF).

Назва мови виникла зовсім не від [виду плазунів](https://uk.wikipedia.org/wiki/%D0%9F%D1%96%D1%82%D0%BE%D0%BD%D0%B8). Автор назвав мову на честь популярного британського комедійного серіалу 70-х років[*«Повітряний цирк Монті Пайтона»*](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%BD%D1%82%D1%96_%D0%9F%D0%B0%D0%B9%D1%82%D0%BE%D0%BD). Втім, все одно назву мови частіше асоціюють саме зі змією, ніж з фільмом — [піктограми](https://uk.wikipedia.org/wiki/%D0%9F%D1%96%D0%BA%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0) файлів в [**KDE**](https://uk.wikipedia.org/wiki/KDE) або в [**Windows**](https://uk.wikipedia.org/wiki/Windows), і навіть емблема на сайті *python.org* зображують зміїну голову.

Наявність дружньої спільноти користувачів, поряд з дизайнерською інтуїцією *Гвідо*, вважається одним з головних факторів успіху **Python**. Розвиток мови відбувається згідно з чітко регламентованими процесами створення, обговорення, відбору та реалізації документів **PEP** (Python Enhancement Proposal) — пропозицій щодо розвитку **Python**.

[3 грудня](https://uk.wikipedia.org/wiki/3_%D0%B3%D1%80%D1%83%D0%B4%D0%BD%D1%8F) [2008](https://uk.wikipedia.org/wiki/2008) року, після тривалого тестування, вийшла перша версія **Python** 3000 (або **Python** 3.0, також використовується скорочена Py3k). У **Python** 3000 усунено багато недоліків архітектури з максимально можливим (але не повним) збереженням сумісності зі старими версіями. На сьогодні підтримуються обидві гілки розвитку (**Python** 3.2 і 2.7).

* 1. ***Бібліотека pygame***
     1. ***Загальна характеристика.***

**pygame** – безкоштовна бібліотека мови програмування Python з відкритим кодом побудована на основі **SDL***(Simple DirectMedia Layer)*. Використовується в основному для створення мультимедійних додатків. Як і **SDL**, pygame володіє високою кросплатформеністю та запускається майже на всіх платформах та операційних системах.

Підтримує Linux (pygame входить до складу найпопулярніших його дистрибутивів), Windows (95, 98, me, 2000, XP, Vista, 64bit windows etc), Windows CE, BeOS, MacOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX, та QNX. Код також може підтримувати AmigaOS, Dreamcast, Atari, AIX, OSF/Tru64, RISC OS, SymbianOS, та OS/2, але це не офіційні розширення.

Також **pygame** не потребує наявності *OpenGL*. Так як *OpenGL* створює більше проблем, ніж вирішує.

Функції ядра бібліотеки написані на **С** та **Assembly**. Це робить бібліотеку ще швидшою, в силу того, що **С** код зазвичай в 10-20 разів швидший за **Python**, а **Assembly в свою чергу в 100 раз швидший, ніж Python.**

Спільнота **pygame –** це невелика група волонтерів, які люблять створювати красиві речі. Як зазначено на офіційному сайті, крім людей до спільноти належать декілька котів, коал, десяток собак, 3.14 гномів і 42 робота.

Вона не претендує на звання найкращої ігрової бібліотеки. Навіть на місце другої кращої. Але розробники вважають, що вона хороша.

* + 1. ***Історія***

**pygame** – була створена *Пітом Шанерсом (Pete Shinners)*, щоб замінити PySDL після того, як розробка останнього була призупинена. Бібліотека була проектом спільноти з 2004 по 2005, і випускалась під **GNU Lesser General Public License** *(Загальна громадська ліцензія обмеженого використання*[*GNU*](https://uk.wikipedia.org/wiki/GNU)*)*.

* 1. ***Допоміжні бібліотеки***
     1. ***Pickle***

**Pickle** – модуль, який наслідує бінарні протоколи, для серіалізації та де серіалізації об’єктів мови **Python**. Входить до складу стандартної бібліотеки. Завдяки цьому модулю можна легко, ефективно та швидко зберігати об’єкти створені користувачем.

* + 1. ***pyinstaller***

**pyinstaller** – модуль, який дозволяє збирати всі необхідні для виконання скрипту бібліотеки в один файл. Це дозволяє уникнути необхідності наявності інтерпретатора **Python** та необхідних бібліотек для кінцевого користувача. Результатом роботи цього модуля буде виконуваний файл, певної операційної системи *(.exe для* **Windows***).*

**Розділ 2.Опис створеного програмного продукту**

* 1. ***Постановка задачі***

У курсовій роботі поставлена задача створити комп’ютерну гру, яка б містила в собі елементи програмування. За основу було взято ідею проходження умовного лабіринту. Його проходження ускладнене ти, що в користувача є обмежена кількість переміщень вперед і назад. Крім того, головний герой гри має напрямок руху, тобто для коректного проходження його треба повертати і рухати вперед чи назад.

Також в гру були добавлені команди початку та кінця двох ітераційного циклу. Тобто всі повороти і переміщення в цьому циклі будуть виконуватися двічі.

Гра потребує:

* логічного мислення
* просторової орієнтації
* розуміння циклів
* передбачливості
* обачливості

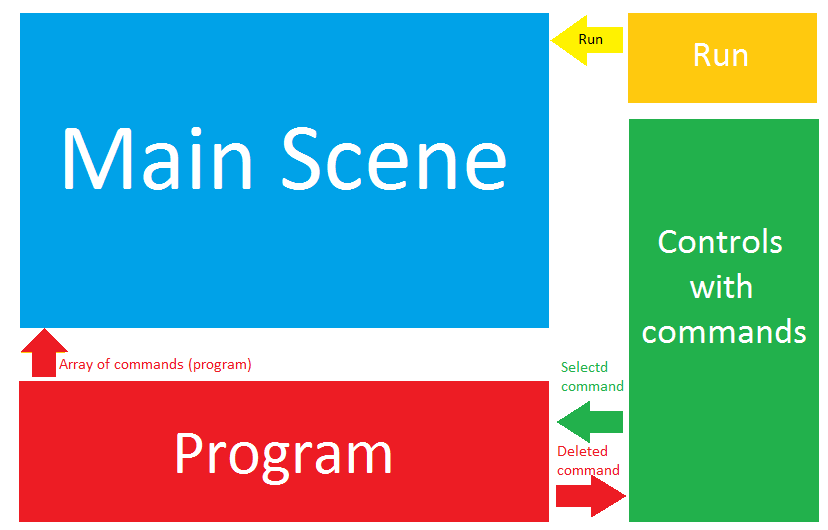
Основні завдання:

* Створити приємний та зручний інтерфейс
* Мультимедійне наповнення гри
* Створити рівні з поясненням
* Портувати гру на популярні платформи
  1. ***Структура проекту***
     1. ***Структура проекту***

Важливим етапом розробки було розділення гри на модулі. Грамотний розподіл в майбутньому облегшує написання програми, та її подальшу підтримку.

З огляду на бажаний графічний інтерфейс були утворені модулі:

* **Команди** – модуль в якому відображаються всі доступні користувачу команди
* **Програми** – модуль із послідовності команд для робота, які були обрані користувачем.
* **Головна** **сцена** – модуль, де відображатиметься самий рівень, і його проходження за програмою користувача.



На рисунку відображені модулі, та обмін інформацією між ним. Модуль **Команди** передає в модуль **Програми** інформацію про натиснуту в ньому команду. Модуль Програми додає в кінець списку цю програму. І навпаки, якщо з модуля Програми було обрано деяку команду, то вона видаляється з нього і передається назад в модуль Команди. При запуску програми користувачем, вся послідовність команд з модуля Програми передається в модуль Сцени.

* + 1. ***Файлова структура***

В силу модульності, проект було вирішено розділити на такі папки:

* **Module** – директорія всіх базових класів.
* **Source** – директорія всіх мультимедійних файлів
* **Level** – директорія з рівнями
* **Libraries** – додаткові функції для особливих потреб
  1. ***Базові модулі***
     1. ***Sprite***

**Sprite** – один із основних примітивів **pygame**. Клас розміщений в *pygame.sprite.Sprite().* Використовується для відображення, збереження, трансформації та взаємодії з графічними об’єктами. Виникла потреба розширити цей функціонал. Тому був написаний клас, який наслідував бібліотеку *pygame.sprite.Sprite().*

**class Sprite**(pygame.sprite.Sprite)**:  
 def** \_\_init\_\_(self, *name***=None**, *placement***=**(0, 0), *s***=**size)**:** *"""  
 :param name: Ім’я спрайта (!!!)  
 :param placement: Розміщення відносно деякої поверхні  
 :param s: Розміри  
 """* pygame.sprite.Sprite.\_\_init\_\_(self)  
 self.name **=** *name* self.image **= None** self.font **= None** self.group **=** pygame.sprite.Group()  
 self.group.add(self)  
 self.rect **=** Rect(*placement*[0], *placement*[1], *s*[0], *s*[1])

В конструктор передається ім’я, розміщення та розмір спрайта.

Додаткові функції:

**def load\_image**(self, *image\_path*, *key***=-**1)**:** *"""  
 Завантажує зображення з директорії та присвоює його спрайту  
 :param image\_path: Шлях до зображення  
 :return:  
 """* self.image, rect **=** image(*image\_path*, *key*)  
 **def set\_font**(self, *some\_font*)**:** *"""  
 Присвоює спрайту деяки шрифт  
 :param some\_font: Об’єкт класу pygame.font  
 :return:  
 """* self.font **=** *some\_font***def set\_name**(self, *some\_name*)**:** *"""  
 Надає спрайту ім’я/ярлик/заголовок і т.п.  
 :param some\_name: Ім’я  
 :return:  
 """* self.name **=** *some\_name***def set\_placement**(self, *point*)**:** *"""  
 Задає позицію відмалювання спрайту на деякій поверхні  
 :param point: Вектор 2Д позиції  
 :return:  
 """* self.rect.x **=** *point*[0]  
 self.rect.y **=** *point*[1]  
  
**def get\_name**(self)**:** *"""  
 :return: Ім’я спрайту  
 """* **return** self.name  
  
**def get\_position**(self)**:** *"""  
 :return: Позицію  
 """* **return** self.rect.x, self.rect.y  
  
**def place\_image**(self, *im*, *place*)**:** *"""  
 Поміщає зображення в певну позицію на спрайті  
 :param im: об’єкт класу pygame.image  
 :param place: місце (x, y)  
 :return:  
 """* surf **=** pygame.Surface((self.rect.width, self.rect.height), pygame.SRCALPHA)  
 surf.blit(*im*, *place*)  
 self.image **=** surf  
  
**def collision**(self, *point*)**:** *"""  
 Перевіряє чи точка знаходиться на спрайті  
 :param point: розміщення  
 :return: булеве значення  
 """* **return** self.rect.collidepoint(*point*)  
  
**def collide\_group**(self, *group*)**:** *"""  
 Перевіряє чи спрайт стикається з групою інших спрайтів  
 :param group: група спрайтів pygame.sprite.Group  
 :return: булеве значення, спрайти із групи з якими зіткнувся  
 """* result **=** []  
 **for** i **in** *group***:  
 if** self.rect.colliderect(i.rect)**:** result.append(i)  
 **if** len(result) **==** 0**:  
 return False**, **None  
 else:  
 return True**, result  
  
**def copy**(self)**:** *"""  
 :return: Копію цього спрайта   
 """* **return** copy.copy(self)  
  
**def blit**(self, *surf*)**:** *"""  
 Відмальовує спрайт на деякій поверхні  
 Extension  
 :param surf: об’єкт класу pygame.surface  
 :return:  
 """* self.group.draw(*surf*)

* + 1. ***Surface***

**Surface** – основний клас **pygame**, на якому відображаються спрайти. На його основі були написані всі вищеперераховані ігрові модулі.

Клас був доповнений:

**class Surface**(pygame.Surface)**:  
 def** \_\_init\_\_(self, *pos***=**position, *s***=**size, *c***=**color, *font***=None**)**:** pygame.Surface.\_\_init\_\_(self, *s*)  
 self.rect **=** pygame.Rect(*pos*[0], *pos*[1], *s*[0], *s*[1])  
 self.group **=** pygame.sprite.Group()  
 self.color **=** *c* self.font **=** *font* self.mouse **= None** self.hover **= None** self.echo **= None** self.update()  
  
 **def update**(self)**:**

*"""  
 Обновлює поверхню (перемальовує і обновлює всі її спрайти)   
 :return:  
 """* **if** self.color **is not None:** self.fill(self.color)  
 self.group.draw(self)  
  
 **def is\_in**(self, *point*)**:** *"""  
 Перевіряє чи деяка точка екрану належить поверхні  
 :param point: tuple  
 :return: Bool  
 """* **return** self.rect.collidepoint(*point*)  
  
 **def on\_surface**(self, *point*)**:** *"""  
 Преводить систему відліку точки на поверхню  
 :param point:  
 :return:  
 """* **return** *point*[0] **-** self.rect.x, *point*[1] **-** self.rect.y  
  
 **def collide\_all**(self, *point*)**:** *"""  
 Зіткнення спрайтів поверхні з точкою  
 :param point: tuple  
 :return: Список зіткнених спрайтів  
 """* result **=** list()  
 point **=** self.on\_surface(*point*)  
 **for** i **in** self.group.sprites()**:  
 if** i.collision(*point*)**:** result.append(i)  
 **return** result  
  
 **def event**(self, *mouse*, *event*)**:** *"""  
 Реація на події  
 :param event: pygame.event  
 :param mouse: pygame.mouse  
 :return: список натиснутих спрайтів в полі self.echo, та зфокусованих в полі self.hover  
 """* self.echo **= None** self.hover **= None  
 if** self.is\_in(*mouse*.get\_pos())**:** self.hover **=** (self.collide\_all(*mouse*.get\_pos()))  
 self.mouse **=** *mouse* **if** *event*.type **==** pygame.MOUSEBUTTONUP **or** self.mouse.get\_pressed()[2]**:** self.echo **=** self.hover  
 self.make()  
  
 **def make**(self)**:**

*"""  
 Функція яка викликається після б-я події на цій поверхні  
 :return: В базовому класі вона пуста  
 """* **pass  
  
 def set\_position**(self, *pos*)**:** *"""  
 Змінює позицію відображення поверхні  
 :param pos:  
 :return:  
 """* self.rect.x **=** *pos*[0]  
 self.rect.y **=** *pos*[1]  
  
 **def set\_font**(self, *some\_font*)**:** *"""  
 Задає шрифт для поверхні  
 :param some\_font: pygame.font  
 :return:  
 """* self.font **=** *some\_font* **def get\_echo**(self)**:** *"""  
 Отримує список натиснутих спрайтів  
 :return: echo any  
 """* self.update()  
 **return** self.echo  
  
 **def echo\_out**(self)**:** *"""  
 Очищає список натиснутих спрайтів  
 :return:  
 """* self.echo **= None  
  
 def flush**(self)**:** *"""  
 Очищає поверхню  
 :return:  
 """* self.group.empty()  
 self.hover **= None** self.echo **= None**

* 1. ***Основні модулі***
     1. ***Command***

До класу **Command** відносяться команди, які користувач може використати в процесі гри. Тобто крок вперед, назад, повороти вправо, вліво, відкриття та закриття двох ітераційного циклу.

За його основу було взято клас **Sprite**, та додано властивості:

* Лічильник – для відображення наявної кількості даної команди
* Ознака заповнення – призначена для зміни вигляду спрайта при наведенні на нього мишкою
* Напрямок – для зручності інтерфейсу, та спрощення орієнтації користувача. Стрілки кроку вперед та назад повертаються разом із створеною користувачем програмою. Таким чином йому легше зрозуміти в яку сторону, на даному етапі створення програми, дивиться головний герой.

Конструктор класу виглядає так:

**class Command**(Sprite)**:  
 def** \_\_init\_\_(self, *name*, *placement***=**(0, 0), *image\_path***=None**, *s***=**size, *countable***=True**)**:** *"""  
 :param name: Назва  
 :param placement: Розміщення  
 :param image\_path: Шлях зображення команди  
 :param s: Розмір  
 """* Sprite.\_\_init\_\_(self, *name*, *placement*, *s*)  
 image\_path **=** *image\_path* **if** *image\_path* **is not None else** def\_location **+** *name* **+** expansion  
 self.load\_image(*image\_path*)  
 self.set\_font(variables.font\_medium)  
 self.big\_font **=** variables.font\_shaded  
 *# Початкове недеформоване зображення* self.original **=** self.image  
 *# Чи для цієї команди встановлений лічильник* self.countable **=** *countable  
 # Напрямок команди* self.direction **=** 0  
 *# Затуханння команди* self.fade **= False** *# Чи вона заповнена курсором* self.hover **= False** *# Початкова кількість доступних команд цього типу* self.amount **=** 0  
 self.update()

Важливою є функція оновлення:

**def update**(self)**:** *"""  
 Оновлює команду  
 :return:  
 """*

*# Створюється тимчасова поверхня* surf **=** pygame.Surface((self.rect.width, self.rect.height), pygame.SRCALPHA)

*# На неї розміщується зображенння команди та повертається на певний кут, в залежності від напряму*  
 surf.blit(pygame.transform.rotate(self.original, self.direction **\* -**90), (0, 0))

*# Якщо в ньому є лічильник*  
 **if** self.countable**:**

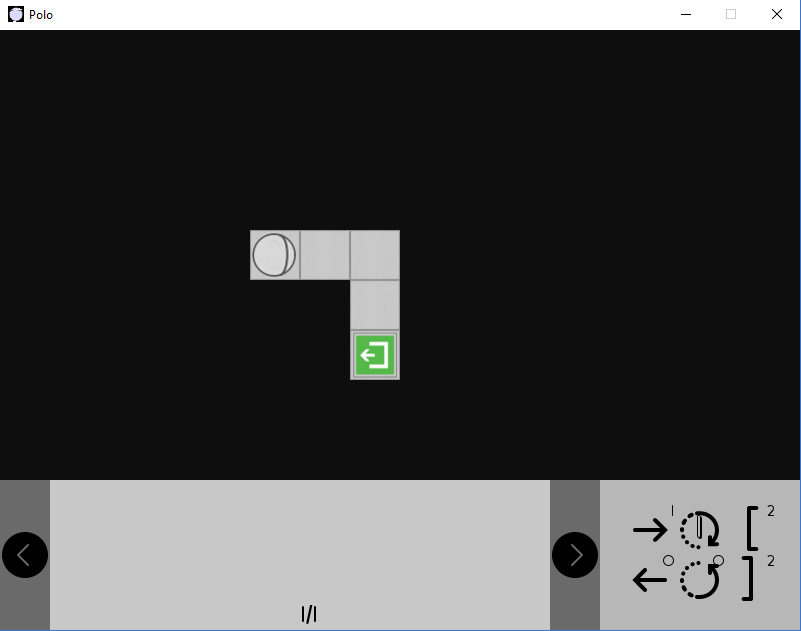
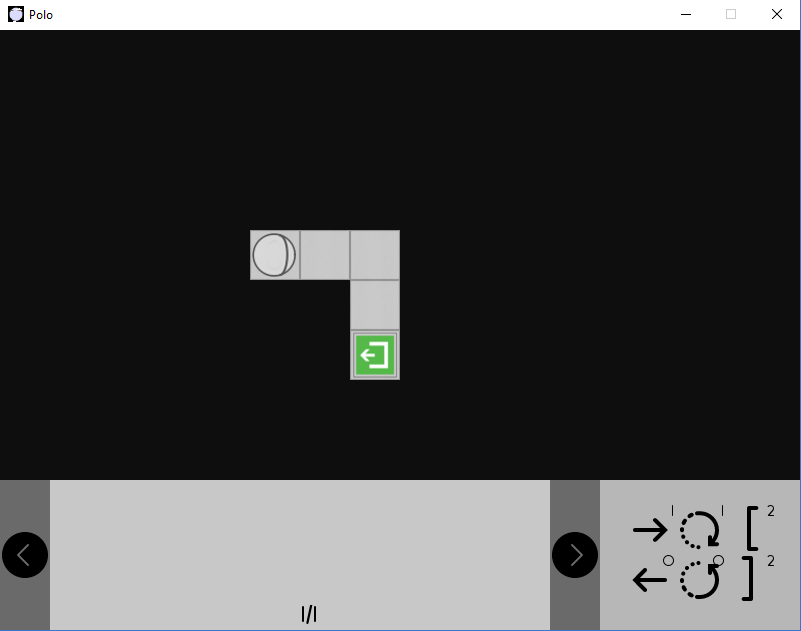
*# Якщо курсор на цій командів* **if** self.hover**:**

*# То розміщуй лічильник великим і по центру*text **=** self.big\_font.render(str(self.amount), 2, (0, 0, 0))  
 text\_size **=** self.big\_font.size(str(self.amount))  
 surf.blit(text, ((self.rect.width **-** text\_size[0]) **/** 2, (self.rect.height **-** text\_size[1]) **/** 2))  
 **else:**

*# Інакше маленьким та в кутку*text **=** self.font.render(str(self.amount), 2, (0, 0, 0))  
 text\_size **=** self.font.size(str(self.amount))  
 surf.blit(text, (self.rect.width **-** text\_size[0], 0))

*# Зображенню спрайта присвоюй те що вийшло на тимчасовій поверхні*  
 self.image **=** surf

Такими маніпуляціями був досягнутий поворот зображення команди та його зміна при наведенні:



* + 1. ***Controls***

Модуль в якому знаходяться доступні користувачу команди. Наслідує клас **Surface**:

**class Controls**(surface.Surface)**:  
 def** \_\_init\_\_(self, *pos***=**position, *s***=**size)**:** self.current\_direction **=** 0  
 self.direction **=** 0  
 *# Елементи як були наведені курсором востаннє* self.was\_hover **= None** *# Створення та розміщення команд* self.forward **=** Command(**"forward"**, (command.width **/** 2, command.height **/** 2))  
 self.back **=** Command(**"back"**, (command.width **/** 2, command.height **/** 2 **+** command.height))  
 self.right **=** Command(**"right"**, (command.width **/** 2 **+** command.width, command.height **/** 2))  
 self.left **=** Command(**"left"**, (command.width **/** 2 **+** command.width, command.height **/** 2 **+** command.height))  
 self.lo **=** Command(**"lo"**, (command.width **/** 2 **+** 2 **\*** command.width, command.height **/** 2))  
 self.op **=** Command(**"op"**, (command.width **/** 2 **+** 2 **\*** command.width, command.height **/** 2 **+** command.height))  
 *# Ініціалізація поверхні* surface.Surface.\_\_init\_\_(self, *pos*, *s*)  
  
 self.group.add(self.forward, self.back, self.left, self.right, self.lo, self.op)

Щоб кожній команді було відомо що відбується і як їй відображатися була перезавантажено функцію **make**:

**def make**(self)**:** *"""  
 Працює з натиснутими спрайтами  
 :return:   
 """  
 # Знімає заповнення з попередньо заповнених елементів і надає його новим* **if** self.was\_hover **is not None:  
 for** i **in** self.was\_hover**:** i.hover **= False** i.update()  
 **if** self.hover **is not None:  
 for** i **in** self.hover**:** i.hover **= True** i.update()  
 self.was\_hover **=** self.hover

*# Зменшує кількість натиснутої команди, якщо це можливо та повертає її копію в self.echo*  
 result **=** []  
 **if** self.echo **is not None:  
 for** i **in** self.echo**:  
 if** i.get\_amount() **>** 0**:** result.append(i)  
 i.change\_amount(**-**1)  
 *# На випадок правого кліку передаються вся доступна кількість* **if** self.mouse.get\_pressed()[2]**:  
 while** i.get\_amount() **>** 0**:** result.append(i)  
 i.change\_amount(**-**1)  
 self.echo **=** result **if** len(result) **>** 0 **else None**

* + 1. ***Program***

Модуль **Program** – відповідає за відображення вже створеної програми. Від модуля **Controls** відрізняється тим, що має сторінки та відображення послідовності спрайтів динамічне.

Були додані спрайти перегортання сторінок, список який містить програму, лічильник поточної сторінки, та їх максимальна кількість.

self.page\_prev **=** Command(**"prev\_page"**, (0, command.height))  
self.page\_next **=** Command(**"next\_page"**, (size[0] **-** command.width, command.height))  
self.listing\_counter **= None**self.program **=** []  
self.page **=** 0  
self.max\_page **=** 0

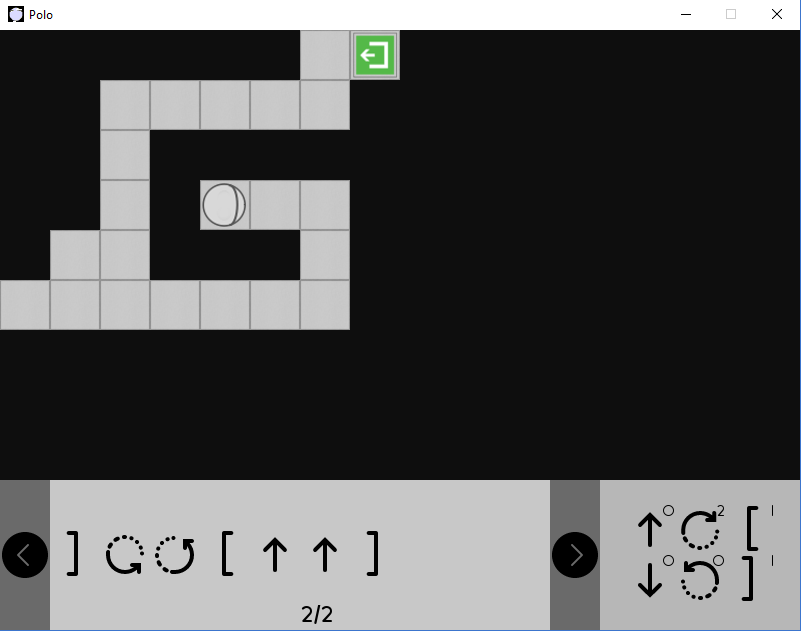
Також для того, щоб на спрайті команди перегортання сторінки не відображалося їхня поточна кількість, для них була викликана функція, яка робить їх «незліченними»:self.page\_next.uncountable()  
self.page\_prev.uncountable()  
self.page\_next.update()  
self.page\_prev.update()

Оновлення модуля відбувається таким чином:

**def update**(self)**:**self.group\_up()  
 self.fill(self.color)  
 pygame.draw.rect(self, sec\_color, (0, 0, command.width, size[1]))  
 pygame.draw.rect(self, sec\_color, (size[0] **-** command.width, 0, size[0], size[1]))  
 self.group.draw(self)  
 self.listing\_update()  
 self.blit(self.listing\_counter, (size[0] **/** 2, size[1] **-** 25))  
  
**def group\_up**(self)**:** *"""  
 Відображає значення, які знаходяться в полі self.program в залежності від поточної сторінки   
 :return: Поточні спрайти на сторінці  
 """* self.group.empty()  
 self.group.add(self.page\_next)  
 self.group.add(self.page\_prev)  
  
 place **=** 0  
 **for** i **in** range(0, len(self.program))**:  
 if** self.page **\*** commands **<=** i **<** (self.page **+** 1) **\*** commands**:** place **+=** command.width  
 self.program[i].set\_placement((place, command.width))  
 self.program[i].update()  
 self.group.add(self.program[i])

*self.listing\_counter* – відображає поточну, та загальну кількість сторінок.

В результаті модуль виглядає так:



* + 1. ***Robot***

**Robot** – клас, який наслідує **Sprite**, та відповідає за відображення і поведінку головного героя гри.

**class Robot**(sprite.Sprite)**:  
 def** \_\_init\_\_(self, *n***=**name, *i\_n***=None**, *placement***=**(50, 50))**:** sprite.Sprite.\_\_init\_\_(self, *n*, *placement*)  
 i\_n **=** *i\_n* **if** *i\_n* **is not None else** def\_location **+** *n* **+** expansion  
 self.load\_image(*i\_n*)  
 self.dead **= False** self.death\_sprites **=** load.load\_sliced\_sprite(def\_location **+** death **+** expansion, width, height)  
 self.death\_sprite\_current **=** 0.0  
 self.death\_sound **=** load.sound(def\_location **+** sound\_explosion)  
 self.death\_sound.set\_volume(sound\_volume)  
 self.original **=** self.image  
 self.fx **=** float(self.rect.x)  
 self.fy **=** float(self.rect.y)  
 self.direction **=** 2

Особливими полями є:

* dead – ознака смерті (випадок виходу головного героя за межі клітинок)
* death\_sprites – послідовність спрайтів, яка анімує смерть.
* death\_sound – звук, якому відповідає смерть
* self.fx, self.fy – позиція робота в типі з плаваючою крапкою.
* direction – напрямок руху робjта (0 – верх, 1 – вправо і т. д.)
  + 1. ***Tile***

**Tile** – клас відповідає за відображення клітинок ігрового поля.

**class Tile**(Sprite)**:  
 def** \_\_init\_\_(self, *name*, *placement***=**(0, 0), *image\_path***=None**)**:** *"""  
 :param name: Ім’я клітинки, або її тип  
 :param placement: Розміщення  
 :param image\_path: Шлях до зображення цієї клітинки  
 """* Sprite.\_\_init\_\_(self, *name*, *placement*)  
 image\_path **=** *image\_path* **if** *image\_path* **is not None else** location **+** *name* **+** expansion  
 self.load\_image(*image\_path*)

* + 1. ***Scene***

Модуль на якому виконується створена користувачем програма, та її результат.

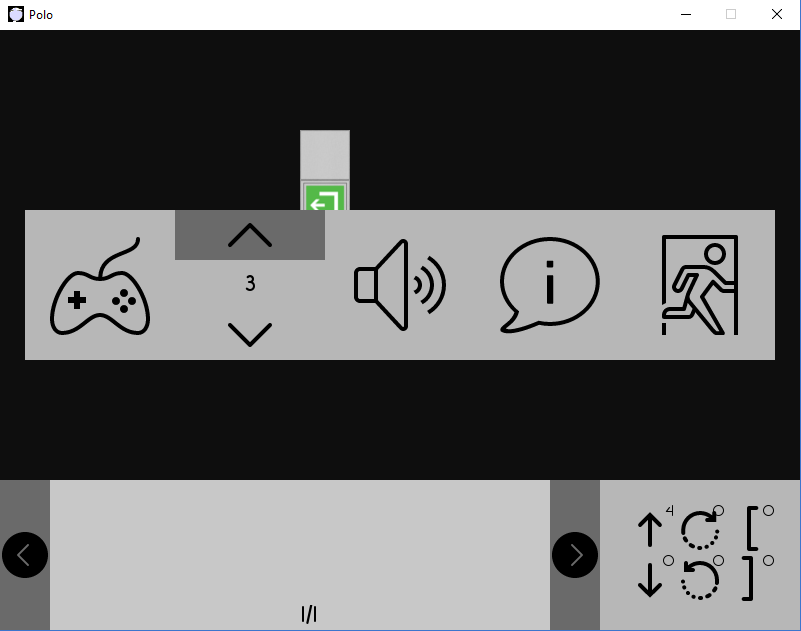
**class Scene**(surface.Surface)**:  
 def** \_\_init\_\_(self, *pos***=**position, *s***=**size, *c***=**color)**:** self.robot **=** Robot()  
 surface.Surface.\_\_init\_\_(self, *pos*, *s*, *c*)  
 *# Фонова музика* self.background\_music **=** load.sound(def\_location **+** background\_music)  
 self.background\_music.play(**-**1)  
 self.background\_music.set\_volume(background\_music\_volume)  
 *# Створена програма* self.program **=** []  
 *# Ознака виконання програми на сцені* self.launch **= False** *# Ознака закінчення виконання* self.done **= False** *# Чи був досягнутий успіх* self.success **= False** *# Смерть робота* self.death **= False** *# Поточна виконувана команда* self.current **= -**1  
 *# Поточний час виконання сцени* self.timing **=** 0  
 *# Опція прискорення виконання сцени* self.speed\_up **= False** self.update()

Для відображення виконання програми головним героєм, виконується функція *step()*, яка залежить від часу, який пройшов від останнього виклику цієї ц функції.

**def step**(self, *tick*)**:** *"""  
 :param tick: Час з останього виклику  
 :return:  
 """  
 # Якщо виникла смерть із функції state()* **if** self.death**:** *# Виконуй дії смерті* self.action(**"die"**, *tick* **\*** speed\_up **\*** speed\_up)**else:** *# Якщо виникла потреба прискорення виконання* **if** self.speed\_up**:** *tick* **\*=** speed\_up  
 self.timing **+=** *tick* **if** self.timing **>=** time**:  
 '''Якщо час випав одночасно на дві дії в програмі'''** self.timing **%=** time  
 self.action(self.program[self.current], *tick* **-** self.timing)  
 self.current **+=** 1  
 **if** self.current **<** len(self.program)**:** self.action(self.program[self.current], self.timing)  
 **else:** self.action(self.program[self.current], *tick*)  
  
 *# Перевірка стану* self.state()

* + 1. ***Menu***

Додатковий модуль меню, який розширює взаємодію користувача та гри.

Містить в собі попередньо описані класи та технології.

Пункти меню дозволяють:

* Продовжити гру
* Вбрати рівень
* Вимкнути фонову музику
* Отримати інформацію про гру
* Вийти з гри

Меню викликається класичною клавішою **ESC**, або при натисканні на клітинку фінішу в модулі **Scene.**

* + 1. ***Message***

Дозволяє надавати тестову інформацію користувачу. В основному це стартові пояснення до цілі та управління гри.

Для мовного різноманіття існує можливість створення підказок/повідомлень на різних мовах. Повідомлення знаходяться в: Levels/мова/назва\_рівня

Якщо рядок починається з «-» це означає, що повідомлення буде відображене на початку рівня. А «+», що після його проходження.

Приклад:

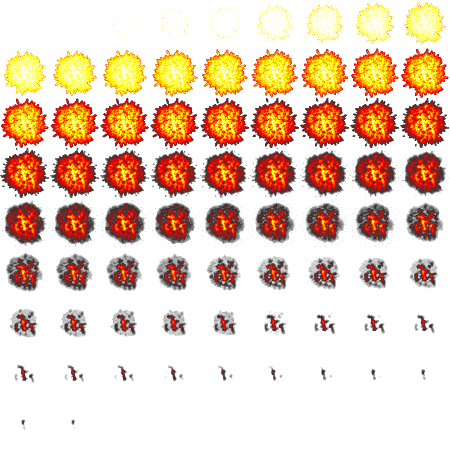


* 1. **Додаткові файли**
     1. ***Load***

В файлі load.py зібрані функції для завантаження мультимедійних файлів гри. Зокрема завантаження зображень для спрайтів, розділених спрайтів та звуків.

Розділений спрайт це послідовність зображень, яка утворює певну анімацію.

До прикладу:



Послідовною зміно цих зображень для спрайта можна досягнути анімації. На зображені – анімація смерті робота.

* + 1. ***Level***

Клас **Level** був створений з метою збереження, зміни та використання створених рівнів. Легким шляхом, було б зберегти за допомогою бібліотеки pickle стан модуля **Scene** та **Controls**, проте, розробниками бібліотеки **pygame**, не було передбачена серіалізація примітивів **Sprite** та **Surface**, на яких оснований модуль **Scene** та **Controls**. Тому виникла потреба створити клас **Level**, який містив би в собі розміщення клітинок, робота, та доступні команди. Ці всі параметри являють собою базові типи мови **Python**, тому їхня серіалізація не створює жодних проблем.

* + 1. ***Creator***

Це скрипт, призначення якого створювати та редагувати рівні гри за командами користувача. Оснований на **CLI**, він є не надто зручним, але робочим редактором рівнів. Створений заради подальшого доповнення гри користувацькими рівнями. До рівнів також можна легко додавати повідомлення для користувача. Цей процес був описаний вище.

* + 1. ***Variables***

**Variables.py** – файл який містить в собі основні константи гри, такі як розширення та режим екрану, кількість кадрів в секунду, мова інтерфейсу, шрифти та їх розмір та інше.

* 1. ***Основний двигун***

Важливим і найскладнішим етапом було відображення та налаштування взаємодії між модулями. Клас Engine містить в собі вище згадані модулі.

**class Engine:  
 def** \_\_init\_\_(self)**:** pygame.init()  
 *# Виклик паузи/меню* self.pause **= False** *# Ознака виклику повідомлення користувачу* self.talk **= False** *# Розміщення вікна* os.environ[**'SDL\_VIDEO\_WINDOW\_POS'**] **= "%d,%d" %** (0, 30)  
 *# Заголовок вікна* pygame.display.set\_caption(window\_title)  
 *# Задається розширення та режим відображення* self.display **=** pygame.display.set\_mode(screen\_resolution) *# Встановлюється іконка* icon **=** load.image(**"Source/Prop/polo.png"**)  
 **if** icon **is not None:** pygame.display.set\_icon(icon[0])  
 *# Встановлення рівня по замовчуванню* self.current\_level **=** 0  
 self.max\_level **=** len(load.get\_levels()) **-** 1  
 *# Ініціалізація годинника* self.clock **=** pygame.time.Clock()  
 *# Завантаження рівня* self.level **=** Level(load.get\_levels()[self.current\_level])  
 *# Звук* self.sound **= True** *# Модулі*

self.controls **=** Controls()  
 self.program **=** Program()  
 self.scene **=** Scene()  
 self.menu **=** Menu()  
 self.message **=** Message()  
 self.voice **=** Voice()

В pygame оновлення робочого вікна повинне викликатися. Тому зазвичай створюється цикл, в якому викликається оновлення вікна.

Але в цьому приховані дві проблеми:

1. Не завжди 200 оновлень вікна за секунду краще, ніж 30. Так як око людини вже при 25 оновленнях вікна в секунду не помічає заміни зображень. Для гри такого типу, частота кадрів несе швидше естетичний характер. Тому було вирішено обмежити кількість оновлень до 60 в секунду.

B **pygame** передбачена функція *pygame.time.Clock.tick(FPS)*, яка приймає обмеження оновлень кадрів. Вона викликається в циклі оновлень. Тобто ця функція перевіряє скільки часу пройшло з останнього її виклику, і штучно «заповільнює» ітерацію, щоб р результаті користувач отримав потрібну кількість оновлень.

Також варто згадати про інший шлях, а саме *pygame.time.delay()* та *pygame.time.wait()*, але їхнє використання може трактуватися деякими операційними системами, як процес завантаження і курсор користувача перетвориться в піктограму очікування. Це може заставити користувача думати, що гра зависла. Тому від таких методів довелось відмовитися.

2. Зайві оновлення. З огляду на тип гри можна сказати, що вона доволі статична, високого рівня динаміки не спостерігається. З цього можна зробити висновок, що при тих чи інших умовах, деякі частини екрану, а саме модулі, не змінюються взагалі. Тобто на їх відображені можна значно «зекономити». В бібліотеці **pygame** передбачені дві функції: *pygame.display.flip()* та *pygame.display.update(rect)*. Перша оновлює все вікно, а друга в свою чергу отримує, як параметр, прямокутник в якому буде відбуватися оновлення. Переваги використання *update(rect)* очевидні. Використання цих двох функцій відображено в функціях:

**def update\_all**(self)**:** *"""  
 Оновлює все.  
 :return:  
 """* self.display.blit(self.controls, self.controls.rect)  
 self.display.blit(self.program, self.program.rect)  
 self.display.blit(self.scene, self.scene.rect)  
  
 **if** self.talk**:** self.display.blit(self.message, self.message.rect)  
 **elif** self.pause**:** self.display.blit(self.menu, self.menu.rect)  
 pygame.display.flip()  
  
**def update**(self)**:** *"""  
 Оновлює модулі, які зазнали змін.*

*Відносно розміщення курсора та стану програми.   
 :return:  
 """* mouse **=** pygame.mouse.get\_pos()  
 **if** self.talk**:** self.display.blit(self.message, self.message.rect)  
 pygame.display.update(self.message.rect)  
 **elif** self.pause**:** self.display.blit(self.menu, self.menu.rect)  
 pygame.display.update(self.menu.rect)  
 **else:  
 if** self.program.is\_in(mouse) **or** self.controls.is\_in(mouse)**:** self.display.blit(self.controls, self.controls.rect)  
 self.display.blit(self.program, self.program.rect)  
 pygame.display.update(self.program.rect)  
 pygame.display.update(self.controls.rect)  
 **if** self.scene.is\_in(mouse) **or** self.scene.launch**:** self.display.blit(self.scene, self.scene.rect)  
 pygame.display.update(self.scene.rect)

За взаємодію між модулями відповідає функція *invoker():*

**def invoker**(self)**:** *"""  
 Модульне спілкування  
 :return:  
 """* **"""Message Echo"""  
 if** self.message.get\_echo() **is not None:** self.message\_handler()  
 self.message.echo\_out()  
  
 **"""Controls Echos"""  
 if** self.controls.get\_echo() **is not None:** self.program.add(self.controls.get\_echo())  
 *# Direct turn* self.controls.set\_delta\_direct(self.program.get\_delta\_direction())  
 self.controls.echo\_out()  
  
 **"""Program Echos"""  
 if** self.program.get\_echo() **is not None:** self.controls.add(self.program.get\_echo())  
 *# Direct turn* self.controls.set\_delta\_direct(self.program.get\_delta\_direction())  
 self.program.echo\_out()  
  
 **"""Scene Echos"""  
 if** self.scene.get\_echo() **is not None:** self.scene\_handler()  
 self.scene.echo\_out()  
  
 **"""Menu Echos"""  
 if** self.menu.get\_echo() **is not None:** self.menu\_handler()  
 self.menu.echo\_out()

**"""Level Changing"""  
 if** self.menu.current\_level **!=** self.current\_level**:** self.menu.set\_current\_level(self.current\_level)  
 self.menu.update()

Решта функцій містять в собі просту і зрозумілу логіку. Тому на їхньому описі увага не загострюється.

**Висновки:**

Важливим аспектом життя, є вибір майбутнього напрямку розвитку. Тому ця гра буде корисною для будь-яких сегментів освітньої системи. Може використовуватись як оцінка потенціалу того, чи іншого індивіда, та також як засіб тренування і розваги. Проте може використовуватися лише як допоміжний засіб навчання.

Одним із головних етапів написання проекту, є його загальний опис та постановка основних задач. Відштовхуючись від цих елементів можна побудувати якісний, легко підтримуваний продукт. Цей етап є визначним для таких мов програмування, як **Python** в міру його строгої динамічної типізації.