МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. І.І. МЕЧНИКОВА

ФАКУЛЬТЕТ МАТЕМАТИКИ, ФІЗИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

**Індивідуальне завдання**

З дисципліни: “Програмування”

на тему:

Створення ієрархії класів на тему «Первісне плем'я»

виконав: Мовчан І.О,

студент 1 курсу факультету

МФІТ

керівник:

канд. ф. -м. н., доц.

Антоненко О. С.

Одеса – 2022

1. **Постановка задачі**

Створити класи: "Людина", а також спеціалізовані класи за видами діяльності - "Збирач", "Рибалка", "Мисливець". У людини передбачити набір характеристик - наприклад, таких як фізична сила, які впливають на успішність його професійної діяльності (для різних спеціалізацій по-різному). Під час виконання відповідні характеристики поліпшуються. Також передбачити наявність умінь, властивих усім людям – як є, спати тощо.

Промоделювати взаємодію людей у первісному племені.

1. **Алгоритм роботи програми**

Користувач має змогу інтерактивно корситуватися програмою. Тобто можна викликати методи в різному порядку та різноманітну кількість рядів. Основні методи реалізовані в класі Human.

Спочатку користувач потрапляє на створення персонажу. Там йому буде запропановано вибрати професію, а потім розподілити очки атрибутів для того щоб зробити персонажу деякі характеристики. Є 3 види атрибутів: сила, спритність та інтелект. Кожен із них допомагає людині при виконанні праці, а саме: сила випромінює опірність до травм, спритність дозволяє уникати їх частіше, а інтелект допомагає отримувати більше видобутку.

Далі користувач може вибрати 5 методів роботи зі своїм персонажем: printAttributes, printHealthInfo, work, eat, sleep та 1 інформаційний метод: help, який виводить усі можливі команди для взаємиодії. Так як продукти додаються на полицю, то і основні методи реалізовані в класі Human.

Якщо у персонажу, не залишилося життів, то людина не може продовжувати роботу з програмою.

1. **Програмна реалізація**

Проєкт налічує в собі 4 класи, а саме:

1. Human,
2. Hunter,
3. Collector,
4. Fisher,

Реалізовано базовий, абстрактний клас Human, який включає в себе наступні поля:

**int** strength;

сила

**int** agility;

спритність

**int** intelligence;

інтелект

**int** remaining\_food;

їжа яка залишилась

**int** earned\_food;

зароблена їжа

**int** hp;

кількість здоров’я

**int** hunger;

голод

**int** fatigue;

втома

**int** upgrade\_points;

кількість очок покращення атрибутів

Конструктори класа Human:

Human();

конструктор без параметрів

Human(int strength, int agility, int intelligence);

конструктор з параметрами

~Human();

деструктор

Основні методи класа Human:

**void** printAttributes() const;

виводить атрибути персонажу

**void** pritnHealthInfo() const;

виводить кількість здоров’я, їжі та втоми персонажу

**int** getStrength() const;

повертає значення поля strength для користування в інших класах

**int** getAgility() const;

повертає значення поля strength для користування в інших класах

**int** getIntelligence() const;

повертає значення поля intelligence для користування в інших класах

**int** getHp() const;

повертає значення поля hp для користування в інших класах

**int** getFood() const;

повертає значення поля remaining\_food для користування в інших класах

**int** getHunger() const;

повертає значення поля hunger для користування в інших класах

**int** getFatigue() const;

повертає значення поля fatigue для користування в інших класах

**int** getUpgradePoints() const;

повертає значення поля upgrade\_points для користування в інших класах

**virtual void** work();

віртуальний метод, реалізований для подальшої реалізації праці разних професій персонажів

**void** eat();

метод, який виконує дію «їсти», тобто додає очки здоров’я, але віднімає кількість залишившиїся їжі

**void** sleep();

метод, який виконує дію «спати», тобто додає очки здоров’я та втоми, але віднімає кількість голоду персонажа

**void** upgradeStrength();

підвищує атрибут «сила» на 1 одиницю

**void** upgradeAgility ();

підвищує атрибут «спритність» на 1 одиницю

**void** upgradeIntelligence ();

підвищує атрибут «інтелект» на 1 одиницю

**void** isDead () const;

перевіряє чи залишились очки здоров’я у персонажа

**void** checkHealth ();

перевіряє чи залишились очки голоду та втоми у персонажу, та якщо вони дорівнюють нулю, віднімає очки здоров’я персонажа

**void** statsRoundUp ();

метод, який перевіряє чи поля не більше 100 та менше 0, та якщо це стається, округляє їх

**virtual void** getInjured ();

метод, який віднімає очки здоров’я персонажу при поранені

Реалізовано клас-спадкоємець Hunter від класу Human, який включає в себе наступні методи:

Конструктори класа Hunter:

Hunter();

конструктор без параметрів

Hunter(int strength, int agility, int intelligence);

конструктор з параметрами

~Hunter();

деструктор

Основні методи класа Hunter:

**void** work () override;

метод «праці», який додає випадкову кількість їжі, в залежності від інтелекту за такою формулою: rand() % intelligence

**void** getInjured () override;

метод, який з шансом rand() % agility <= 2 віднімає очки здоров’я персонажу при поранені за такою формулою hp -= 70 / strength;

Реалізовано клас-спадкоємець Collector від класу Human, який включає в себе наступні методи:

Конструктори класа Collector:

Collector();

конструктор без параметрів

Collector(int strength, int agility, int intelligence);

конструктор з параметрами

~Collector();

деструктор

Основні методи класа Collector:

**void** work () override;

метод «праці», який додає випадкову кількість їжі, в залежності від інтелекту за такою формулою:

earned\_food = rand() % intelligence + 1;

if (earned\_food == intelligence - 1) earned\_food = intelligence - 2;

**void** getInjured () override;

метод, який з шансом rand() % agility <= 1 віднімає очки здоров’я персонажу при поранені за такою формулою hp -= 45 / strength;

Реалізовано клас-спадкоємець Fisher від класу Human, який включає в себе наступні методи:

Конструктори класа Fisher:

Fisher();

конструктор без параметрів

Fisher(int strength, int agility, int intelligence);

конструктор з параметрами

~ Fisher();

деструктор

Основні методи класа Fisher:

**void** work () override;

метод «праці», який додає випадкову кількість їжі, в залежності від інтелекту за такою формулою:

earned\_food = rand() % intelligence;

if (earned\_food == intelligence - 1) earned\_food = intelligence - 3;

**void** getInjured () override;

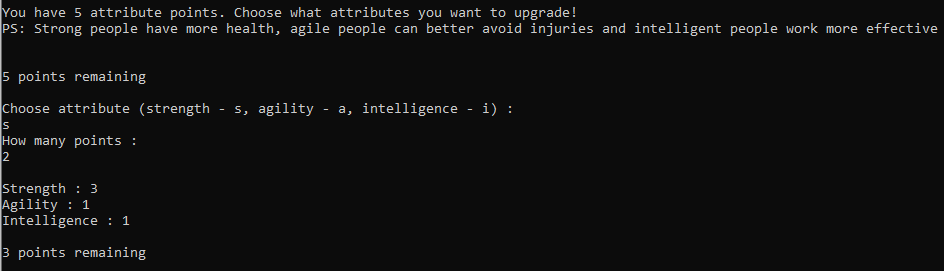
метод, який з шансом rand() % agility <= 1 віднімає очки здоров’я персонажу при поранені за такою формулою hp -= 30 / strength;

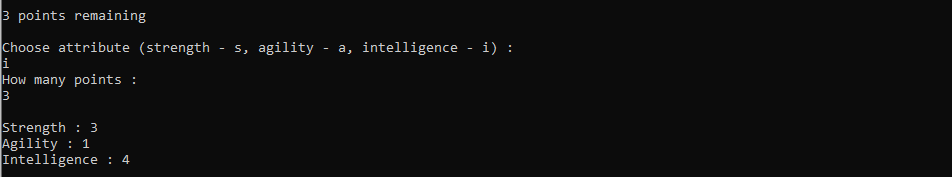
1. **Демонстрація програми**
2. **Створення персонажу**

Спочатку вам буде запропановано вибрати спеціалізацію персонажа, отже давайте оберемо, наприклад, мисливця:



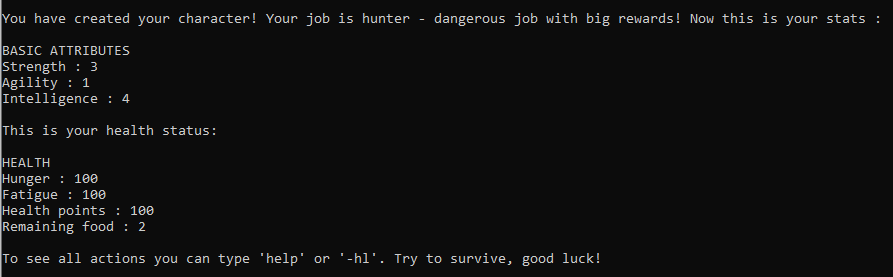
Після цього буде дано 5 очок атрибутів, які ми можемо витратити на покращення значень атрибутів персонажу, від яких буде залежати наскільки успішно пермонаж зможе виживати. Наприклад, покращимо силу на 2 та інтелект на 3, ігноруючи спритність:



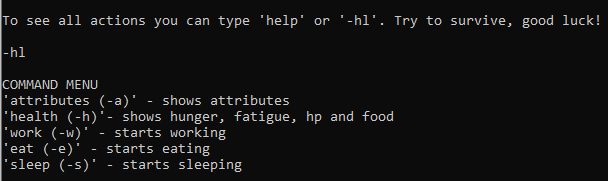


1. **Основна інформація персонажа**

Отже, після створення персонажу, користувачу буде показано атрибути та очки життєдіяльності, які йому потрібно буде витрачати обережно, оскільки на початку персонажеві вижити буде доволі важко:



Пропоную вам написати команду –hl та побачити перелік основних команд:

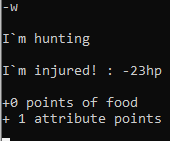


Тут ми побачимо такі команди, як:

* attributes (-a) – вивести перелік атрибутів (метод getAttributes() )
* health (-h) – вивести очки життєдіяльності (метод getHealthInfo() )
* work (-w) – почати «працювати» (метод work() )
* eat (-e) – почати «їсти» (метод eat() )
* sleep (-s) – почати «спати» (метод sleep() )

1. **Основна програма**

Нарешті спробуємо «попрацювати», тоді напишемо –w у поле введення:

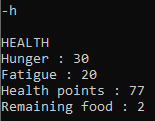


Можемо побачити, що після введення –w почалося виконання программи, тобто:

* Персонаж почав «працювати» (I`m hunting)
* Після «праці», персонаж травмувався (I`m injured)
* Почав заробив 0 очок їжі (+0 points of food)
* Персонаж отримав 1 очко атрибутів (+1 attribute points)

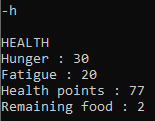
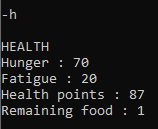
На цей раз персонажу не повезло, оскільки він не заробив їжі та травмувався. Отже він втратив очки здоров’я, втоми та голоду, які ми можемо побачити ввівши команду –h (вивести очки життєдіяльності):

*БУЛО: СТАЛО:*

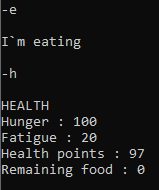


Отже можна відновити очки голоду ввівши команду –e (їсти):

*БУЛО: СТАЛО:*



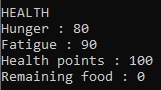
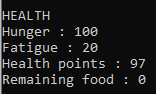
Тобто додалося 40 очок голоду, 10 очок здоров’я та віднялося 1 очко їжі. Повторимо цю дію:



Тепер нам потрібно відновити очки втоми, це ми можемо зробити ввівши команду –s (спати):

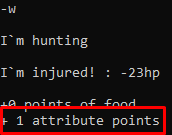


*БУЛО: СТАЛО:*

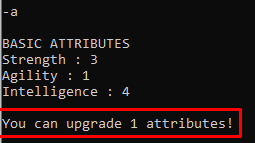


Тобто додалося 70 очок втоми та 10 очків здоров’я, але оскільки максимум = 100, то додалося тільки 3 очки.

Також після праці нам додалося 1 очко атрибутів, про що ми можемо дізнатися після виконання команди –w (працювати) :



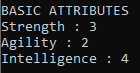
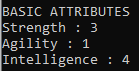
Або після виконання команди –a (вивести очки атрибутів):



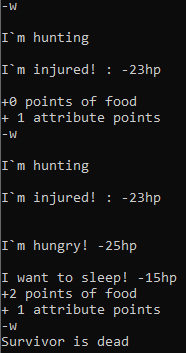
Отже пропоную вам підвищити очко атрибута «спритність». Тоді введемо команду –u (підвищити очко атрибута) та оберемо спритність ввівши “a”:



*БУЛО: СТАЛО:*

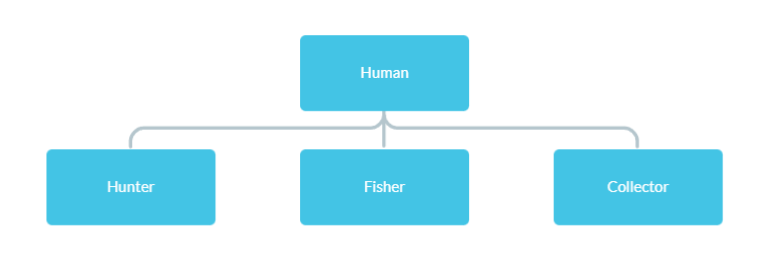


Отже ми протестували усі команди! Залишилося тільки протестувати що буде, якщо здоров’я персонажу опуститься до нуля:



Бачимо що після того як здоров’я персонажу стало дорвнювати 0, то персонаж, загинув та ми більше не можемо користуватися програмою:



1. **Ієрархія класів**
2. **Висновки**

Під час виконання даної роботи, я познайомився з принципами об'єктно-орієнтованого програмування (ООП) на мові програмування С++. Зокрема, я ретельно вивчив поняття ієрархії класів та їх взаємодію між собою.

Для написання правильних методів було використано різні алгоритми. Одним з ключових аспектів у цій роботі було активне використання генераторів псевдовипадкових чисел, оскільки потрібно було генерувати шанси тих чи інших випадкових результатів.

Отже, ця курсова робота чітко продемонструвала мені сутність об'єктно-орієнтованого програмування та його роботу в практиці.