



Софийски университет „Св. Климент Охридски“
Факултет по математика и информатика

*Курсова работа по Обектно-ориентирано програмиране
специалност Компютърни науки*

Тема №8 Джурасик парк

Съдържание

1. Увод.....	3
1.1. Описание и идея на проекта	
1.2. Цели и задачи на разработката	
1.3. Връзка към хранилището в Github	
2. Общ поглед върху проекта.....	4
2.1. Основни дефиниции, концепции и алгоритми, използвани в проекта	
2.2. Подходи и методи за решаване на поставените проблеми	
3. Проектиране	
3.1. Архитектура.....	5
3.2. Диаграми (най-важните извадки от кода).....	8
4. Реализация и тестване.....	10
4.1. Управление на паметта и алгоритми	
4.2. Тестове	
5. Заключение.....	11
5.1. Бъдещо развитие и усъвършенстване	

1. Увод

1.1. Описание и идея на проекта

Проектът „Джурасик парк“ реализира виртуален зоопарк за динозаври, в който потребителят може да създаде различни по брой и вид клетки за виртуалните праисторически влечуги. Идеята е да се напише система за управление на така наречения зоопарк, която да строи клетки, добавя животни, наема хора и зарежда храна за динозаврите.

1.2. Цели и задачи на разработката

Целта на проекта е да се създаде оптимална програма с максимално разнообразна функционалност, която да отговаря на поставената задача.

Проектът е изграден съгласно добрите принципи на ООП – проектиран е така, че да работи с „абстрактни данни“ – данни с неясно представяне, което дава възможност за по-лесно описание и модификация на програмата.

1.3. Връзка към хранилището в Github: https://github.com/Spacepanda21/Jurassic_Park

2. Общ поглед върху проекта

2.1. Основни дефиниции, концепции и алгоритми, използвани в проекта

Данните в проекта са разпределени в съответни класове. За по-голяма достъпност, улеснение и подредба проектът е съставен от *header* и *cpp файлове*, като във всеки *header* се съдържа отделен клас, обединяващ необходимите за неговата имплементация член-данни и член-функции, които са разпределени в *public* и *private* секциите в зависимост от необходимия достъп до тях, а в *cpp файла* се намират дефинициите на декларираните в *header* член-функции. Също така са използвани външни функции, които не са методи на класовете. Използвана е конвенцията за именуване на функции и променливи *snake_case*, като функциите и променливите започват с малка буква, а класовете с главна.

2.2. Подходи и методи за решаване на поставените проблеми

Класовете са изградени по така нареченото *Rule of Three*, т.е. съдържат член-функциите *Destructor*, *Copy Constructor* и *Copy Assignment Operator* и *Constructor*, за да може програмата да работи правилно. В конструктора са инициализирани член-данните чрез обобщена синтактична конструкция (в така наречените инициализиращи списъци), т.е. в заглавието преди изпълнението на тялото на конструктора. Понеже имаме обекти, които не могат да се създадат чрез директно присвояване на член-данните, сме дефинирали копирателен конструктор. Тъй като при изготвянето на проекта не са създавани класове с общи компоненти и поведение като тези на вече дефиниран клас, то не се е налагало използването на идеята на наследяването.

В програмата е използван *enum class* за член-данните на класа динозавър (*Dinosaur*) ера (*era_t ::= TRIASSIC | JURASSIC | CRETACEOUS_PERIODS*), категория (*category_t ::= HERBIVORE | CARNIVORE | PTEROSAUR | SPINOSAUR*) и пол (*sex_t ::= FEMALE | MALE*) и за член-данните на класа храна (*Food*) тип храна (*food_t ::= PLANTS | MEAT | FISH*) понеже според заданието те могат да бъдат само тези няколко определени вида и нищо друго.

Използвани са следните библиотеки:

- `<fstream>` // библиотека за работа с файлове
- `<iostream>` // стандартната библиотека за вход и изход
- `<sstream>` // `stringstream` за функцията `split_string()`;
- `<string>`
- `<time.h>` // `std::rand()`; за генериране на произволно число
- `<vector>`

3. Проектиране

3.1. Архитектура

За имплементацията на проекта са създадени следните класове:

- Dinosaur
- Cage
- Dimensions
- Zoo

и структури:

- Food
- Parser

❖ Клас Dinosaur се характеризира с член-данните в *private* секцията :

- `string name` (името на динозавъра)
- `sex_t sex` (полът на динозавъра)
- `category_t category` (разред на динозавъра)
- `era_t era` (ерата, към която принадлежи динозавърът)
- `string climate` (климатът, който обитава)
- `string type` (видът на динозавъра)
- `food_t food` (храна, с която се храни)

В *public* секцията се намират необходимите конструктор, деструктор, копи конструктор, оператор=, съответните селектори, които да позволят достъп до член-данните в *private* секцията на класа, и метода `print()`; който с подходящи *log* извежда информацията за динозаврите в конзолата.

❖ Клас Cage се характеризира със следните член-данни в секцията със спецификатор за достъп *private*:

- `size_t capacity` // максималният капацитет на клетката
- `Dimensions dimensions` //размерите на клетката са обект от клас `Dimensions`
- `vector<Dinosaur> dinosaurs` //вектор от динозаврите в клетката
- `category_t category` } /*тези три член-данни служат за
- `era_t era` } свързване на динозавъра с
- `string climate` } подходяща клетка*/

Методите на този клас са съответно „Голямата 4-ка“, съответните селектори за достъп до `category`, `era` и `climate`, `print()`; и `list_dinosaurs()`; извеждащи съответната информация за клетката и динозаврите в нея. Също така са дефинирани функциите `void add_new_dinosaur(Dinosaur);` и `void remove_dinosaur(string name);`, които ще помогнат при реализирането на `void add_dinosaur(Dinosaur);` и `void remove_dinosaur_by_name();` в класа `Zoo`. Освен това има и помощни функции като `bool has_free_space();` и `int dinosaur_count();`, които ще бъдат извикани в други функции.

- ❖ Клас **Dimensions** е имплементиран, за да можем да създадем обект от този тип в клас **Cage**, тъй като разглеждаме клетката в тримерното пространство. Съответно член-данните му **x**, **y**, **z** са от тип **double** и служат за означаване на размерите на клетката в пространството. В **public** секцията има съответните конструктор, деструктор, копи конструктор и оператор= както и **void print()**; метода.

- ❖ Клас **Zoo** се състои от член-данните в **private** секцията:

- **vector<Cage> cages** //вектор от клетки в зоопарка
- **Food plants**
- **Food meat**
- **Food fish**
- **int staff_count** //имаме нужда от екип от хора, който да се грижи за динозаврите в зоопарка

и методите:

- **bool is_food_category_compatible(food_t, category_t);** използва се за проверка при подаване на храна на динозавъра, понеже растителноядните динозаври не могат да ядат месо; ако е *false* се извежда подходящо съобщение при създаването на динозавър
- **Dinosaur load_dinosaur(ifstream&);** помощна функция, която ще се използва при четене на информация от файл

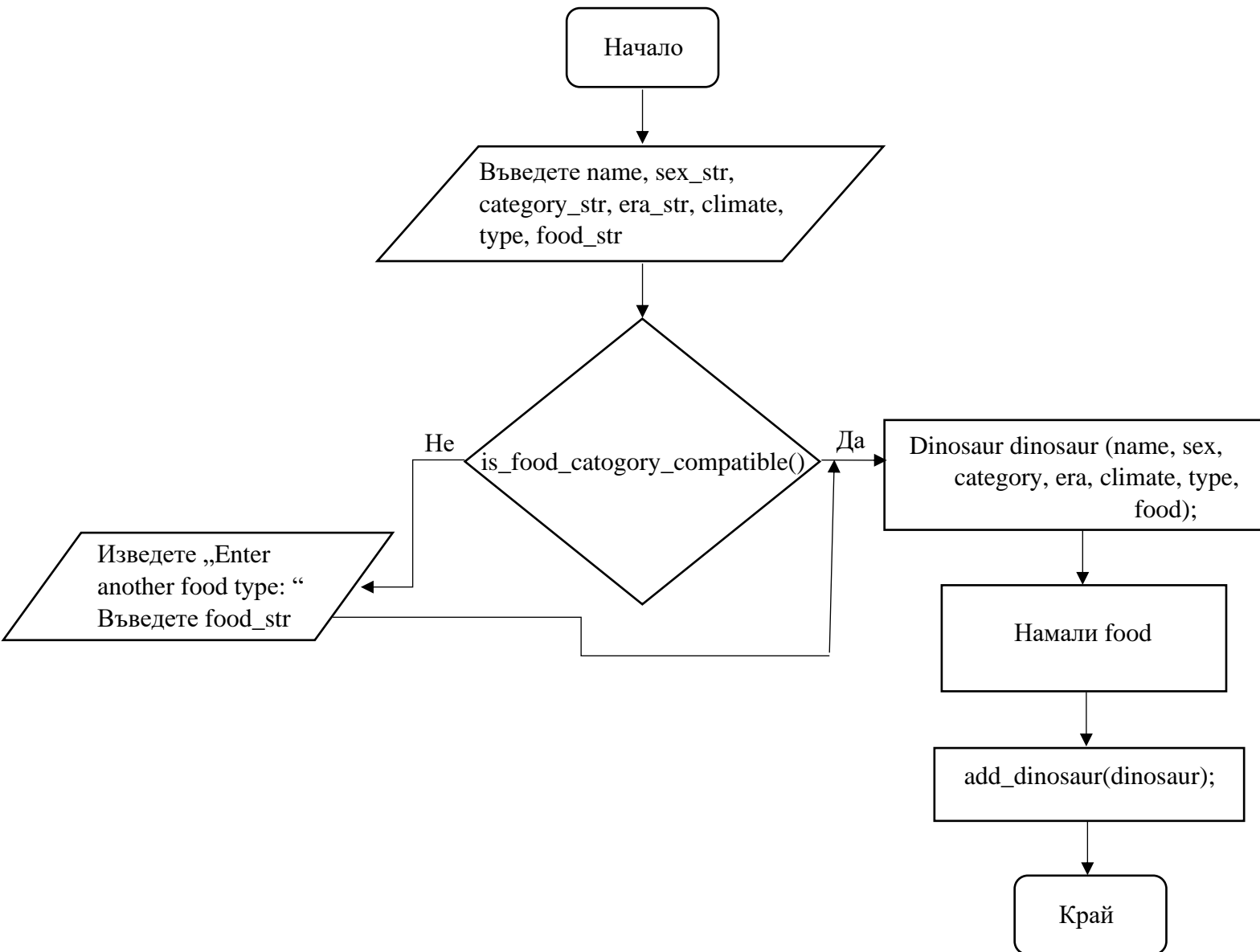
Функции в **public** секцията са:

- Конструктор **Zoo()**; , в който се извиква функцията **void load_from_file(string filename);**, която със създаване на обект от класа **Zoo** започва четене на информацията от файла със съответното име, подадено като аргумент на функцията, както и по-долу описаната функция **void create_random_cages()**;
- Деструктор **~Zoo()**; , в който се извиква функцията **void save_in_file(string filename);** , която с приключването на програмата запазва информацията във файла, подаден като аргумент на метода
- **void load_command();** и **void save_command();** са аналогични на функциите, извиквани в конструктора и деструктора, но позволяват на потребителя да запази информацията във въведен от него файл, защото все пак не искаме въведената от потребителя информация да се губи, каквото е и едно от условията на задачата
- **void create_cage();** и **void create_random_cages();**, чрез които се създават клетки в зоопарка, като разликата е, че методът **void create_random_cages();** се използва за генериране на прозволени брой клетки при стартирането на програмата, каквото е и изискването на задачата, а **void create_cage();** позволява на потребителя да създаде клетка с въведени от него в конзолата параметри
- **void list(ostream&) const;** изкарва в конзолата информация за всички налични клетки в зоопарка – техните параметри и динозаврите вътре

- `void hire_staff();` служи за добавяне на нов служител, което е необходимо, когато броят на динозаврите спрямо хората се увеличи с повече от 3, т.е. 1 човек се грижи за не повече от 3 динозавъра; нуждата от тази функция се подсказва на потребителя при добавяне на динозавър в зоопарка
- `void deliver_food_in_storage();` се използва, за да се зареди определено количество храна в хранилището, като потребителят избира от кой тип и какво количество храна да въведе; необходимостта от тази функция се подсказва на потребителя с подходящо съобщение при добавяне на динозавър в зоопарка
- ❖ Структура **Food** пази храната и количеството ѝ от тип `double` като член-данни и съдържа метода `void print();`
- ❖ Структура **Parser** е необходима при интерпретирането на данните от тип `string`, които са въведени от потребителя, към съответно тип `category_t`, `era_t`, `sex_t`, `food_t` за правилното функциониране на програмата, а ако въведеният от потребителя низ не съответства на никоя от възможните стойности, които приемат съответните `enum class`, се извежда подходящо съобщение; данните ѝ са статични, за да може да има достъп до тях отвсякъде
- ❖ **Main.cpp** съдържа външните функции `void static print_help();`, която функция извежда на конзолата помощно меню, показващо на потребителя кои функции се поддържат от програмата и как трябва да бъдат въведени от него със съответните параметри, и функцията `void run_program(Zoo& zoo);`, приемаща аргумент по референция обект от клас `Zoo` и състояща се от последователност от условни оператори `if`, които проверяват дали въведената от потребителя команда съответства с поддържаните от програмата функции; ако съвпадат, се извежда подходящо съобщение и се извиква съответната функция. Освен това има и направени тестове, за да се покаже функционалността на програмата.

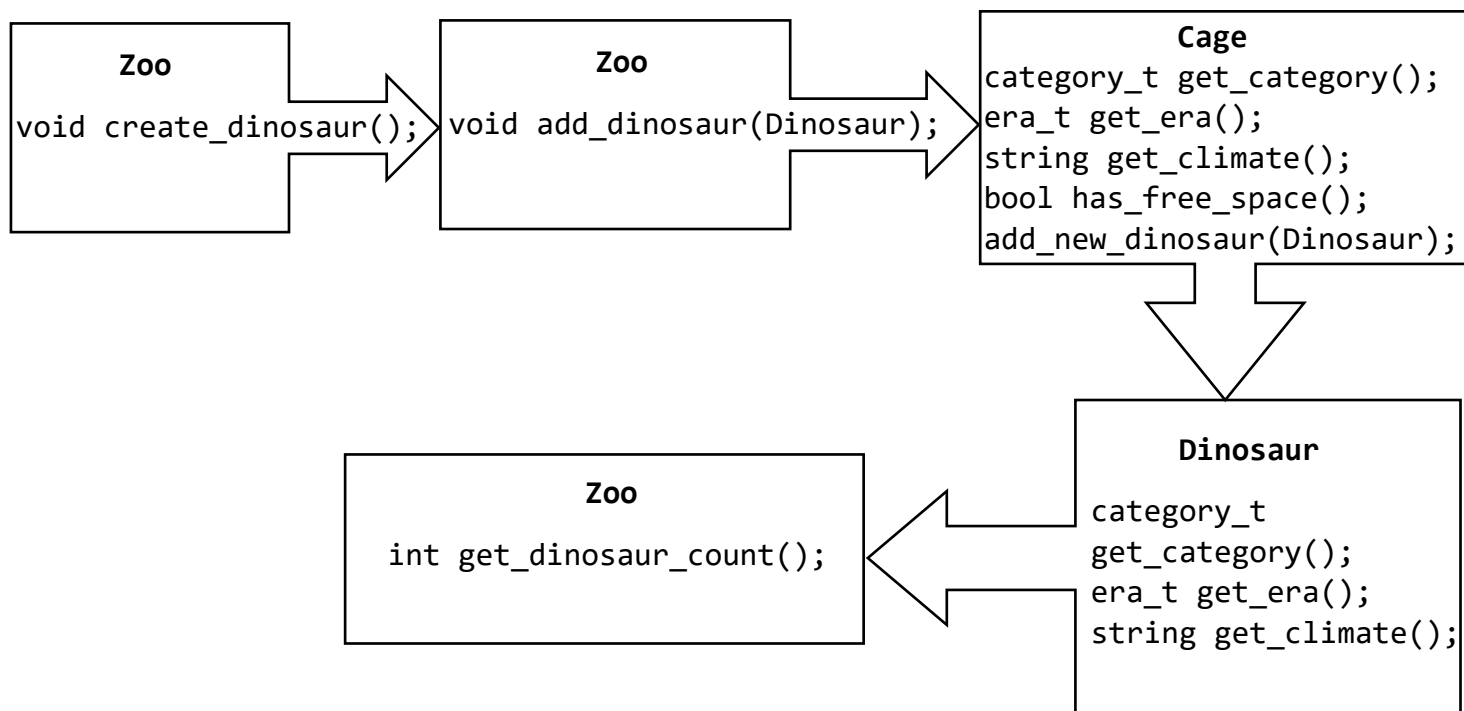
3.2. Диаграми

Фигура 3.2.1 представлява блок схема на функцията `void create_dinosaur()` в класа `Zoo`.



Фигура 3.2.1

На фигура 3.2.2 може да се проследи обобщеното изпълнение на функциите, свързани със създаването на динозавър в различните класове.



Фигура 3.2.2

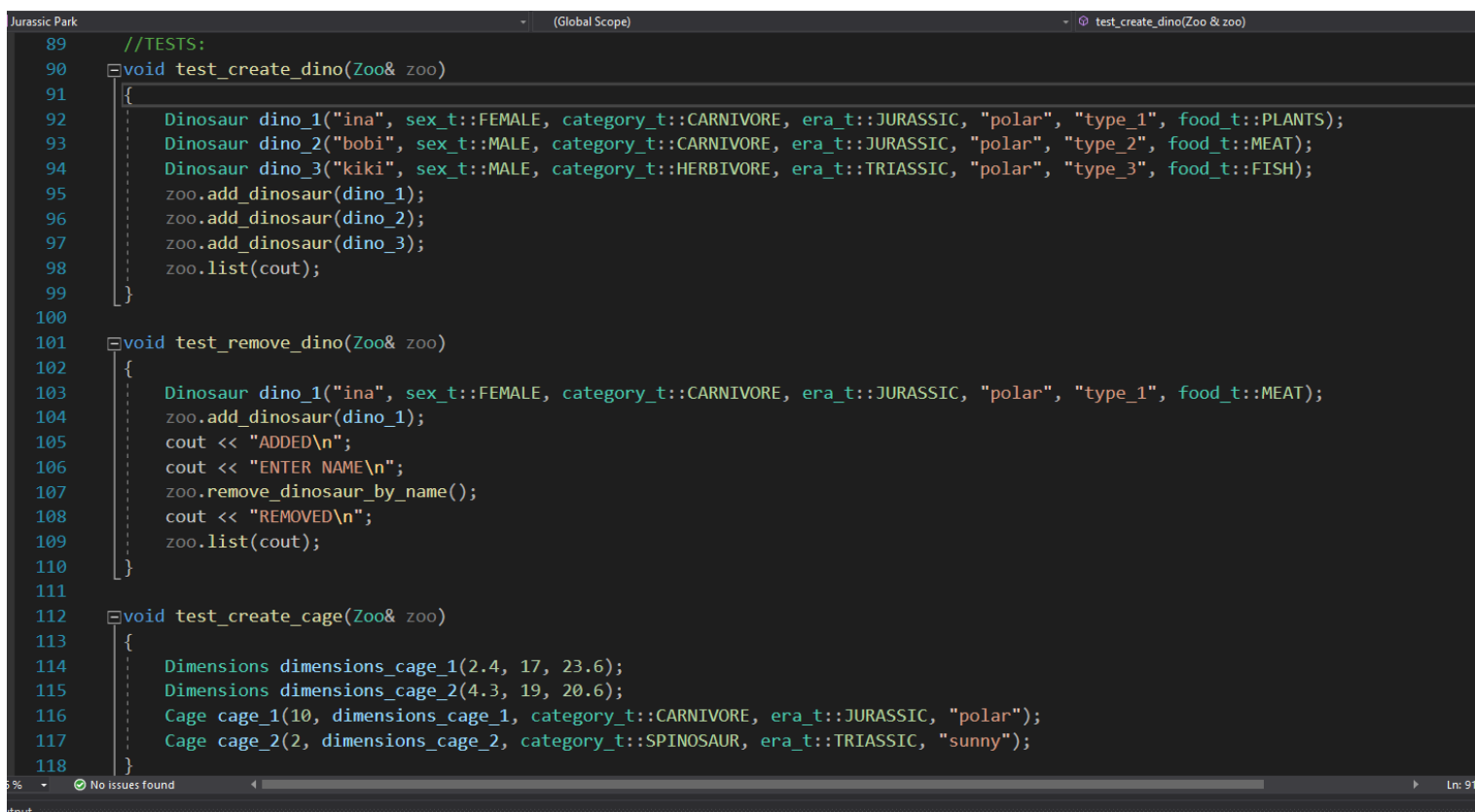
4. Реализация и тестване

4.1. Управление на паметта и алгоритми

Навсякъде паметта автоматично се заделя и унищожава от класа *vector* в стандартната библиотека, т.е. работим с динамична памет. Използваните алгоритми са с линейна сложност.

4.2. Тестване

При изготвянето на проекта в Main.cpp са направени и необходимите тестове (фигура 4.2.1), които да проверят правилното функциониране на функциите и на цялостната програма.



```
89 //TESTS:
90 void test_create_dino(Zoo& zoo)
91 {
92     Dinosaur dino_1("ina", sex_t::FEMALE, category_t::CARNIVORE, era_t::JURASSIC, "polar", "type_1", food_t::PLANTS);
93     Dinosaur dino_2("bobi", sex_t::MALE, category_t::CARNIVORE, era_t::JURASSIC, "polar", "type_2", food_t::MEAT);
94     Dinosaur dino_3("kiki", sex_t::MALE, category_t::HERBIVORE, era_t::TRIASSIC, "polar", "type_3", food_t::FISH);
95     zoo.add_dinosaur(dino_1);
96     zoo.add_dinosaur(dino_2);
97     zoo.add_dinosaur(dino_3);
98     zoo.list(cout);
99 }
100
101 void test_remove_dino(Zoo& zoo)
102 {
103     Dinosaur dino_1("ina", sex_t::FEMALE, category_t::CARNIVORE, era_t::JURASSIC, "polar", "type_1", food_t::MEAT);
104     zoo.add_dinosaur(dino_1);
105     cout << "ADDED\n";
106     cout << "ENTER NAME\n";
107     zoo.remove_dinosaur_by_name();
108     cout << "REMOVED\n";
109     zoo.list(cout);
110 }
111
112 void test_create_cage(Zoo& zoo)
113 {
114     Dimensions dimensions_cage_1(2.4, 17, 23.6);
115     Dimensions dimensions_cage_2(4.3, 19, 20.6);
116     Cage cage_1(10, dimensions_cage_1, category_t::CARNIVORE, era_t::JURASSIC, "polar");
117     Cage cage_2(2, dimensions_cage_2, category_t::SPINOSAUR, era_t::TRIASSIC, "sunny");
118 }
```

Фигура 4.2.1

5. Заключение

5.1. Бъдещо развитие и усъвършенстване

С подходяща графика, звукови ефекти и разширение на функционалността този проект може да се разрасне и да се превърне в забавна онлайн игра за деца. В последствие може да бъде направено и приложение за телефон, за да бъде още по-достъпно.