Учреждение образования

«Белорусский Государственные университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

«Программное средство с использованием интерпретатора кода (Windows)»

по дисциплине

«Системное программное обеспечение вычислительных машин»

Выполнил: Руководитель:

студент гр. 850505

Пикиреня Павел Иванович Глоба А. А.

Минск, 2020

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc41486929)

[ОБЗОР ИСТОЧНИКОВ 4](#_Toc41486930)

[1 СТРУКТУРНОЕ ПРОЕКТИРОВАНИE 5](#_Toc41486931)

[2 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 7](#_Toc41486932)

[3 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 12](#_Toc41486933)

[4 ТЕСТИРОВАНИЕ ПРОГРАММЫ И РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 14](#_Toc41486934)

[5 ЗАКЛЮЧЕНИЕ 21](#_Toc41486935)

[6 СПИСОК ИСТОЧНИКОВ 22](#_Toc41486936)

[ПРИЛОЖЕНИЯ 23](#_Toc41486937)

# **ВВЕДЕНИЕ**

Целью работы является создание языка программирования и рабочего интерпретатора для него.

В данной работе планируется реализация виртуальной машины для управления «роботом» на игровом поле посредством интерпретации и выполнения исходного кода.

Основной недостаток интерпретируемых языков (медленность по сравнению с компилируемыми [2]) в данном случае не имеет значения, так как для удобного восприятия пользователем не имеет смысла выполнять более нескольких команд в секунду.

Также планируется процедурная генерация игрового поля для «робота», для избегания ситуации, в которой игрок создает код, специально подогнанный под конкретную ситуацию. Например – полностью линейная последовательность действий, которая перестает работать при минимальном изменении поля.

# **ОБЗОР ИСТОЧНИКОВ**

Из аналогов можно язык Python [4] – высокоуровневый интерпретируемый язык программирования. Python является open source проектом. Он поддерживает модули, заранее написанные и скомпилированные на более низкоуровневых языках программирования, за счет чего сочетает простоту написания кода интерпретируемых языком и скорость выполнения компилируемых.

Для разработки была использована IDE Visual Studio Community 2019 [5]. Она была выбрана благодаря наличию опыта работы с ней и поддержки системы контроля версий git. Разработка проводилась с использованием ISO C++17 [1].

Для взаимодействия с консолью был выбран Console API [3], так как он значительно превосходит по скорости вывод с стандартный поток вывода и позволяет изменять символы в заданном положении на экране, не перерисовывая при этом весь экран. Это позволяет избежать эффекта «мерцания» при частом обновлении картинки и значительно упрощает вывод данных из разных модулей.

# **СТРУКТУРНОЕ ПРОЕКТИРОВАНИE**

* 1. Общее структурное описание состава программного обеспечения

Данная программа состоит из четырех модулей: управляющий модуль, модуль окружения, модуль игры, модуль вывода на экран.

Управляющий модуль осуществляет инициализацию и последующее управление оставшимися модулями.

Модуль окружения осуществляет исполнение пользовательского кода и, следовательно, управление модулем игры.

Модуль игры отвечает за контроль игрового поля.

Модуль вывода на экран отвечает за показ информации пользователю.

Схема структурная представлена в приложении 4.

* 1. Описание деления проекта на отдельные программы, модули или другие составные части

Управляющий модуль получает имя файла исходного кода как аргумент командной строки и передает его для инициализации модуля окружения. Если имя файла не было передано, программа завершается. После инициализации модуля окружения, управляющий модуль в бесконечном цикле дает модулю окружения команду для выполнения следующей строки кода и выводит выполненную строку на экран. Цикл продолжается до получения первого исключения, в случае чего на экран выводится сообщение от модуля окружения и программа ожидает ввод пользователя для завершения.

Модуль окружения имеет следующие поля: форматированный исходный код, модуль игры, переменные пользовательского кода, условные точки для перехода в пользовательском коде, стек адресов возврата из функций, стек переменных, ссылки на функции в пользовательском коде, указатель на выполняемую команду. После получения названия файла исходного кода модуль загружает исходный код в форматированном виде. После загрузки исходного кода модуль обрабатывает его для нахождения адресов функций, условных точек перехода и переменных. Данные о них заносятся в соответствующие поля модуля. Также указатель на выполняемую команду инициализируется адресом точки входа. На этом инициализация заканчивается. В дальнейшем при получении команды о выполнении строки исходного кода она обрабатывается и выполняются соответствующие действия по изменению переменных, стеков, указателя на команду и флагов состояния. Подробное описание поведения модуля окружения будет описано в разделе «Руководство пользователя».

Модуль игры имеет следующие поля: положение робота, состояние робота, счет робота, информация о игровом поле. Информация об игровом поле состоит из 8-битных значений, являющимися побитовым объединением текущих состояний клетки. При инициализации игра создает поле. В левый верхний угол помещает робота, в правый нижний – цель. Также на поле помещается некоторое число еды. По запросу от модуля окружения модуль игры может представить следующую информацию: текущий счет игрока, текущее положение робота, информация по конкретной клетке, информация о всех клетках, параметры игрового поля. Также модуль окружения может давать запрос на перемещение робота и получать обратную связь в виде состояния робота. Подробное описание правил игры будет дано в разделе «Руководство пользователя».

Модуль вывода на экран представлен в виде класса со статическими методами вывода на экран. Он включает следующие методы: вывод на экран игрового поля через std::cout, вывод на экран игрового поля посредством Console API, вывод на экран сообщения посредством Console API. При первом вызове функции, использующей Console API, выполняется получение дескриптора консоли. Дальнейшая работа ведется через него.

* 1. Краткое описание сторонних программных компонент

Для вывода на экран использовалось Console API [3]. Также были использованы следующие коллекции из стандартной библиотеки: std::vector, std::string, std::stack, std::unordered\_map, std::pair [1].

# **ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

* 1. Управляющий модуль

try

{

runtime sandbox{ filename };

while (true)

{

drawer::write(sandbox.execute(), i++, 40);

if (i >= 24)

{

i = 1;

}

}

}

catch (...)

{

drawer::write(last\_error, 24);

}

Задача управляющего модуля сводится к инициализации модуля окружения и поэтапному вызову метода execute() с последующим выводом на экран. В случае получения исключения, управляющий модуль завершает программу.

* 1. Модуль окружения
     1. Класс runtime

private:

typedef std::unordered\_map<std::string, variable> variables;

typedef std::unordered\_map<std::string, int> points;

script m\_script;

game m\_game;

variables m\_global;

points m\_points;

std::stack<int> ret\_adresses;

std::stack<variable> m\_stack;

std::unordered\_map<std::string, int> functions;

int ip;

public:

runtime(const std::string& filename);

std::string execute();

private:

int find(const std::string& source, const std::string& substr);

int getType(const std::string& str);

std::string getName(const std::string& str);

std::string getWord(const std::string& str, size\_t ind);

variable get\_var(const std::string& str);

Класс runtime отвечает за исполнение кода пользователя.

В поле m\_script хранится исходный код.

В поле m\_game хранится текущая игра.

В поле m\_global хранятся переменные.

В поле m\_points хранятся точки перехода.

В поле ret\_adresses хранится стек точек возврата.

В поле m\_stack хранится стек значений.

В поле functions хранятся адреса функций.

В поле ip хранится указатель на текущую комманду.

Конструктор принимает название файла как аргумент и инициализирует им поле m\_script. Также происходит инициализация поля m\_game. Далее конструктор проходит по строкам исходного кода и если видит объявление переменной, функции или точки перехода заносит информацию о них в соответствующее поле.

Метод execute выполняет текущую строку исходного кода и возвращает строку, состоящую из номера выполненной строки исходного кода и самой строки исходного кода. Если дальнейшее выполнение программы невозможно или не имеет смысла метод заполняет переменную last\_error и генерирует исключение std::exception.

Метод find находит первое вхождение substr в str и возвращает индекс начала этого вхождения. Если вхождения нет, возвращает -1.

Метод getType возвращает тип команды исходного кода.

Метод getName является оберткой для метода getWord, если надо найти второе слово и сгенерировать исключение если оно не является последним.

Метод getWord возвращает слово с индексом ind в строке. Если такого слова нет, возвращает пустую строку. Возвращенная строка никогда не будет содержать символ ‘ ’ и ‘:’.

Метод get\_var возвращает переменную, инициализированную значением str. Метод инициализирует ее числом, если строка str число и переменной, если str не число. Если такой переменной нет, метод генерирует исключение. Метод осуществляет проверку на специальные и инициализирует соответствующим значением переменную.

* + 1. Класс script

public:

std::vector<std::string> source;

script(std::string filename);

Класс script отвечает за хранение исходного кода.

В поле source храниться отформатированный исходный код.

Конструктор построчно сканирует заданный файл и сохраняет символы, находящиеся в белом списке. При этом если пробелов между словами больше одного, записывается только один. Если строка в результате удаления запрещенных символов окажется пустой, она пропускается.

* + 1. Класс variable

private:

union

{

int int\_value;

float float\_value;

} value;

var\_types type;

public:

variable();

variable(int value);

variable(float value);

variable(const variable& other);

variable& operator=(const variable& other);

variable operator\*(const variable& other);

variable operator+(const variable& other);

variable operator-(const variable& other);

variable operator/(const variable& other);

variable operator%(const variable& other);

bool operator==(const variable& other);

bool operator>(const variable& other);

bool operator!=(const variable& other);

bool operator<(const variable& other);

bool operator>=(const variable& other);

bool operator<=(const variable& other);

operator int ();

void reinit(int val);

void reinit(float val);

void reinit(const variable& other);

friend std::ostream& operator<<(std::ostream& out, const variable& val);

Класс variable отвечает за представление значения переменной внутри виртуальной машины.

Поле type существует для определения того, какое из полей объединения имеет смысл. Его значение определяется в зависимости от вызванного конструктора.

Оператор присваивания изменяет значение переменной с сохранением ее типа.

Арифметические операторы создают новую переменную, тип которой определяется согласно правилам языка C++. Но если один из участников оператора none, то результатом будет none.

Операторы сравнение производят сравнения по правилам языка C++. Но если один из участников none, то результат всегда false.

Методы reinit используются для изменения значения переменной, с возможным изменением типа.

Оператор вывода в поток выводит имеющее смысл значение или “none” в поток.

* 1. Модуль игры

#define X first

#define Y second

std::pair<int, int> player\_pos;

state m\_state;

std::vector<std::vector<uint8\_t>> field;

int score;

public:

game(int width, int height);

state moveUp();

state moveDown();

state moveLeft();

state moveRight();

int getScore();

int playerX();

int playerY();

int width();

int height();

tile getTile(int x, int y);

const std::vector<std::vector<uint8\_t>>& getState();

Класс game отвечает за взаимодействие с игрой

В поле player\_pos хранится положение робота.

В поле m\_state хранится состояние робота.

В поле field хранится состояние поля.

В поле score хранится текущий счет игрока.

Конструктор принимает 2 параметра – желаемы высота и ширина поля. В случае, если они выходят за разрешенные границы, параметры приводятся к ближайшей границе. Далее создается поле требуемого размера и инициализируется пустыми клетками, в поле помещается робот и его координаты сохраняются, в поле помещается точка назначения, в поле помещается некоторое количество еды.

Методы, начинающиеся с move, отвечают за перемещения робота на 1 клетку в соответствующем направлении. Если движение невозможно, робот остается на месте. Если на новой клетке находится еда, то счет увеличивается на 1, а еда уничтожается. Если на новой клетке находится точка назначения, то состояние робота изменяется на состояние победы. По окончанию метода возвращается состояние робота.

Метод getScore возвращает текущий счет игрока.

Методы playerX и playerY возвращают соответствующие координаты робота.

Методы width и height возвращают ширину и высоту поля.

Метод getTile возвращает приоритетное значение клетки в данной координате. Если координата выходи за границы поля, клетка считается стеной. Приоритет возвраты выглядит следующим образом (в порядке убывания): робот, точка назначения, еда, пусто.

Метод getState возвращает информацию о поле. Значение каждой клетки представляется как побитовое объединение ее состояний.

* 1. Модуль вывода на экран

public:

static void print(const std::vector<std::vector<uint8\_t>>& field);

static void draw(const std::vector<std::vector<uint8\_t>>& field);

static void write(const std::string& message, int str\_ind, int col\_ind = 0);

Метод print отвечает за вывод на экран игрового поля посредством std::cout.

Метод draw отвечает за вывод на экран игрового поля посредством Console API.

Метод write отвечает за вывод на экран строки посредством Console API.

# **РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

Для примера рассмотрим алгоритм обработки команды, если она является командой умножения, деления, сложение или вычитания. Блок-схема приведена для случая, когда при компиляции не был указан флаг RUNTIME\_DEBUG.



Максимальный размер стека в Cour 1.0 составляет 256 значений.

Верным числом аргументов считается два аргументов.

Инициализация будет успешной, если оба аргумента являются либо легальным числовым значением или именем существующей переменной.

Искусственное ограничение на размер стека было введено для поощрения создания алгоритмов, не рассчитывающих на кешированиe больших объемов данных и уравнивания возможностей пользователей с разным аппаратным обеспечением.

# **ТЕСТИРОВАНИЕ ПРОГРАММЫ И РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

* 1. Руководство пользователя

Для запуска исходного кода на языке Cour название файла передается как аргумент командной строки интерпретатору.

* + 1. Правила игры

Целью игры является достижение роботом точки назначения, получив при этом максимальный счет. Счет увеличивается за счет сбора условной еды. Робот может передвигаться в любую сторону на 1 клетку. Сбор еды и окончание игры происходит автоматически. При попытке выйти за поле робот останется на месте. Если за одну команду требуется обработать несколько аргументов они начинают обрабатываться слева направо.

* + 1. Общая структура языка

Cour является интерпретируемым языком со статической типизацией. Не поддерживает объектно-ориентируемую парадигму. Линейный порядок выполнения кода может нарушаться операторами переходов и вызова функции. Выполнение кода начинается с функции main. При ее отсутствии возможность запуска кода отсутсвует. Комментарии должны начинаться с символа ‘;’. Все имена и названия в языке Cour являются регистрозависимыми.

* + 1. Переменные

Переменные имеют глобальную область видимости и инициализируются перед запуском программы. В языке Cour существует 2 типа переменные: int и float. Тип конкретной переменной определяется при ее объявлении. Имя переменной может включать прописные и заглавные буквы латинского алфавита, цифры и символ ‘\_’. Также переменные нельзя называть именами особых переменных. Пределы переменных соответствуют пределам аналогичных в языке C++.

* + - 1. Особые переменные

Язык имеет несколько заранее объявленных переменных. Они указаны в таблице ниже:

|  |  |
| --- | --- |
| Название | Поведение |
| stack | Принимает значение равное значению на вершине стека без извлечения из стека |
| stackp | Принимает значение равное значение на вершине стека и вынимает верхний элемент из стека |
| none | Пустая переменная |
| player\_x | Текущее значение координаты x робота |
| player\_y | Текущее значение координаты y робота |
| field\_width | Ширина игрового поля |
| field\_height | Высота игрового поля |
| wall | Значение клетки-стены |
| nothing | Значение клетки-пустоты |
| player | Значение клетки-робота |
| food | Значение клетки-еды |
| destination | Значение клетки-точки назначения |

* + 1. Арифметические операции

Арифметические операции представлены в Cour пятью операторам: mul – умножение, div – деление, add – сложение, sub – вычитание, rem – остаток. Все эти операторы подчиняются единому синтаксису: ОПЕРАТОР АРГ1 АРГ2. В качестве аргументов могут выступать любые переменные, в том числе особые, а также числа. Результат арифметической операции заносится на вершину стека. Тип переменной результата определяется по принципу: если хоть одни из операторов был пустой переменной, то результат будет пустой переменной; если хоть один из операторов был float, то результат будет float; в остальных случаях результат – int. Арифметические операции не осуществляют проверки на переполнение переменных и деление на 0. Ответственность за это лежит на программисте. Проверка осуществляется только на целочисленность аргументов в операторе rem.

* + 1. Взаимодействие со стеком данных

В процессе написания программы на Cour программисту придется взаимодействовать со стеком данных. Размер стека данных ограничен. В Cour 1.0 максимальный размер стека данных – 256 переменных. Для помещения значения на вершину стека используется оператор push. Для изъятия значения из стека используется оператор pop. У операторов push и pop единый синтаксис: ОПЕРАТОР АРГ1. При этом аргументом для push может служить любая переменная, в том числе особая, а также число. А аргументом для pop может служить только переменная, объявленная программистом или none. При попытке изъять значение из пустого стека или поместить значение в полный стек выполнение программы останавливается.

* + 1. Условные и безусловные переходы

Для контроля порядка выполнения кода существуют безусловные и условные переходы. Они переносят выполнение программы на определенную метку. Метки определяются перед запуском программы. Метка может включать прописные и заглавные буквы латинского алфавита, цифры и символ ‘\_’. Метка должна оканчиваться символом ‘:’. Условные переходы используют флаги, выставленные оператором сравнения cmp. Синтаксис этого оператора: ОПЕРАТОР АРГ1 АРГ2. В качестве аргументов могут выступать любые переменные, в том числе особые, а также числа. Синтаксис операторов перехода общий для всех операторов перехода: ОПЕРАТОР МЕТКА. Все операторы перехода приведены в таблице ниже:

|  |  |
| --- | --- |
| ja | Выполнить переход если АРГ1 > АРГ2 |
| jb | Выполнить переход если АРГ1 < АРГ2 |
| je | Выполнить переход если АРГ1 = АРГ2 |
| jmp | Выполнить переход |

* + 1. Функции

Функции объявляются оператором proc. Синтаксис оператора: ОПЕРАТОР ИМЯ. Имя функции может включать прописные и заглавные буквы латинского алфавита, цифры и символ ‘\_’. Программа должна содержать как минимум одну функцию под названием main. Дополнительные функции не обязательны. Для вызова функции используется оператор call. Синтаксис оператора совпадает с синтаксисом оператора proc. При вызове, оператор call сохраняет в стек адресов точку возврата и передает управление кодом по адресу функции. Для возврата из функции используется оператор ret. Он передает управление по адресу, лежащему на вершине стека адресов, и забирает значение из стека адресов. Если ret вызывается, когда стек адресов пуст, программа завершается. Ответственность за наличие оператора ret в конце функции лежит на программисте и не контролируется. Если оператор call был вызван, когда стек адресов полон, программа завершается. В Cour 1.0 стек адресов имеет размер 256 адресов.

* + 1. Взаимодействие с игрой

Для взаимодействия с игровым миром есть несколько операторов. Они представлены в таблице ниже:

|  |  |  |
| --- | --- | --- |
| Оператор | Синтаксис | Описание |
| moveUp | ОПЕРАТОР | Передвинуть робота вверх |
| moveDown | ОПЕРАТОР | Передвинуть робота вниз |
| moveLeft | ОПЕРАТОР | Передвинуть робота влево |
| moveRight | ОПЕРАТОР | Передвинуть робота вправо |
| map | ОПЕРАТОР АРГ1 АРГ2 | Занести на вершину стека значение клетки поля по координатам АРГ1 АРГ2 (x y). В качестве аргументов могут выступать любые переменные, в том числе особые, а также числа. Если желаемая точка выходит за пределы поля, на вершину стека занесется wall. |

В клетке может находится сразу несколько свойств. В таком случае возвращается самое приоритетное. Приоритеты следующие (в порядке убывания): робот, точка назначения, еда, ничего.

* 1. Тестирование

Для тестирования была написана простая программа для сбора роботом всей еды и перехода к выходу:

int dir

int x

int y

int has\_pos

proc main

main\_loop:

call get\_dir

cmp dir 1

je move\_up

cmp dir 2

je move\_down

cmp dir 3

je move\_left

cmp dir 4

je move\_right

move\_up:

moveUp

cmp 1 1

je main\_loop

move\_down:

moveDown

cmp 1 1

je main\_loop

move\_left:

moveLeft

cmp 1 1

je main\_loop

move\_right:

moveRight

jmp main\_loop

ret

proc get\_dir

get\_dir\_start:

call get\_target

cmp x player\_x

je get\_dir\_y

jb get\_dir\_left

ja get\_dir\_right

get\_dir\_left:

push 3

pop dir

ret

get\_dir\_right:

push 4

pop dir

ret

get\_dir\_y:

cmp y player\_y

ja get\_dir\_down

jb get\_dir\_up

push 0

pop has\_pos

jmp get\_dir\_start

get\_dir\_up:

push 1

pop dir

ret

get\_dir\_down:

push 2

pop dir

ret

ret

int has\_inited

proc get\_target

cmp has\_pos 1

je get\_target\_end

push 0

pop has\_inited

int buf

sub field\_width 1

get\_target\_loop\_x:

push stack

pop buf

sub field\_height 1

get\_target\_loop\_y:

map buf stack

cmp stack food

je get\_target\_food

cmp stackp destination

ja get\_target\_nothing

jb get\_target\_nothing

get\_target\_dest:

pop y

pop x

push x

push y

push 1

pop has\_pos

jmp get\_target\_nothing

get\_target\_food:

pop none; remove tile data

pop y

pop x

push 1

pop has\_pos

ret

get\_target\_nothing:

sub stackp 1

cmp stack 0

ja get\_target\_loop\_y

je get\_target\_loop\_y

pop none; remove curr y coord

sub stackp 1

cmp stack 0

ja get\_target\_loop\_x

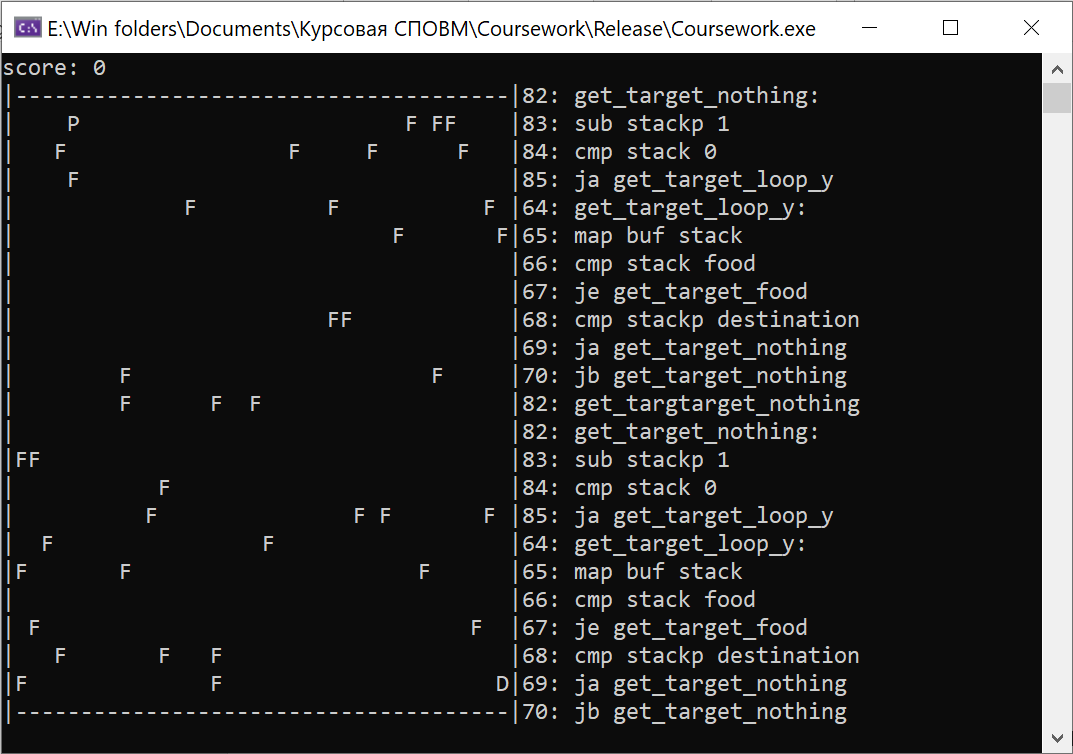
je get\_target\_loop\_x

pop none; remove curr x coord

get\_target\_end:

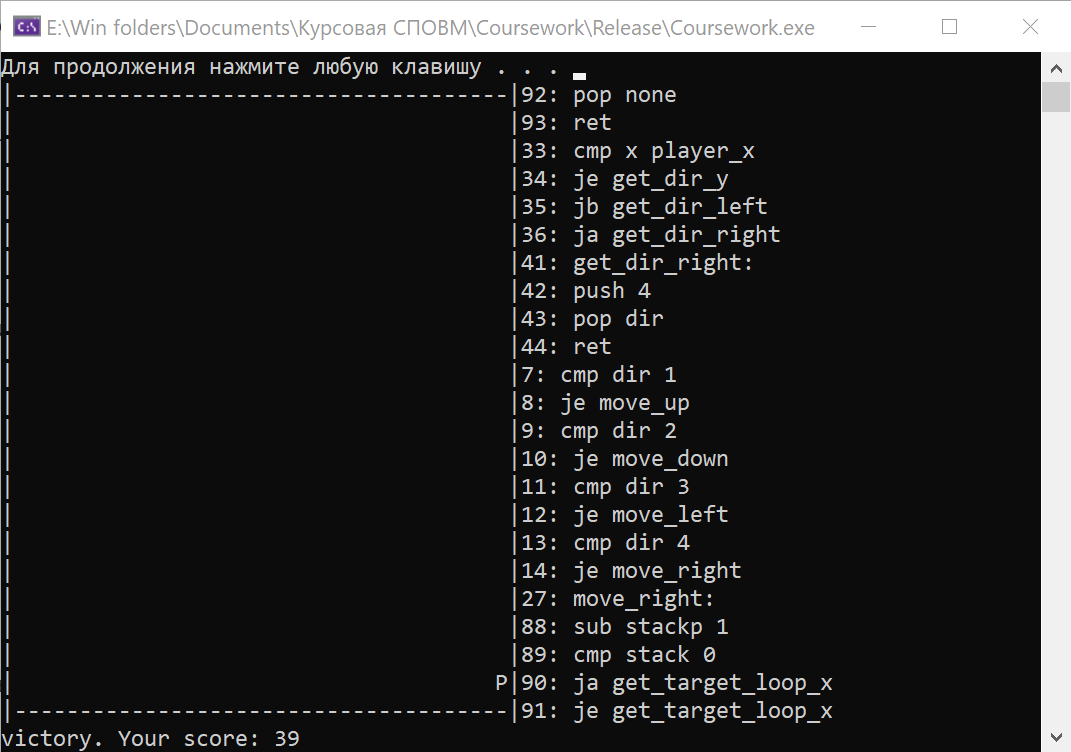
ret

Данный код сохранен в файл code.txt. В интерпретатор передано название файла и начато исполнение:



На экране отображаются игровое поле, текущий счет и быстро сменяющиеся выполняемые команды.

Экран при завершении программы:



# **ЗАКЛЮЧЕНИЕ**

В результате работы над этим проектом была разработана спецификация языка Cour, был создан интерпретатор для него. Был реализована игра, для контроля управления которой и создавался язык Cour. В игре была реализована процедурная генерация поля. Были выполнены все условия, поставленные темой курсового проекта. Был написан пример кода на языке Cour и протестирован. В результате тестирования проблем выявлено не было. Был получен опыт по работе с системой контроля версий git и по оформлению документации к проектам. Главным недостатком по сравнению с существующими языками программирования является невозможность взаимодействия с пользователем.

# **СПИСОК ИСТОЧНИКОВ**

1. ISO C++17 [Электронный ресурс]. – электронные данные. – Режим доступа: <https://en.cppreference.com/w/cpp/17>
2. Дорот В.Л. Толковый словарь современной компьютерной лексики / В.Л. Дорот – 3-е изд. – СПб.: БХВ-Петербург, 2004. – 608с.
3. Low-Level Console I/O [Электронный ресурс]. – электронные данные. – Режим доступа: <https://docs.microsoft.com/en-us/windows/console/low-level-console-i-o>
4. Python [Электронный ресурс]. – электронные данные. – Режим доступа: <https://python.org>
5. Visual Studio Community [Электронный ресурс]. – электронные данные. – Режим доступа: <https://visualstudio.microsoft.com/vs/community/>

# ПРИЛОЖЕНИЯ

Приложение 1

Приложение 2

Приложение 3

Приложение 4