### Описание алгоритмов

### Описание структур данных

Далее представлены основные классы, используемые в части, отвечающей за ход искусственного интеллекта.

1. Класс Checker

Класс представляет собой структуру хранения для шашки.

class Checker

{

int Color; //цвет шашки, 0-белый или 1-черный

int Type; //тип шашки, 0-шашка или 1-дамка

int Coord; //координаты шашки (3 бита – координата по x, 3 бита – координата по y)

int Num; //идентификатор шашки

int NextNum; //номер следующей шашки, для реализации списка шашек на массиве[[1]](#footnote-2)

int PrevNum; //номер предыдущей шашки

public:

Checker(): Color(0), Type(0), Coord(0), Num(0), NextNum(0), PrevNum(0) { }

~Checker() {}

void SetColor(int color);

void SetType(int type);

void SetCoord(int coord);

void SetNum(int num); //устанавливается 1 раз

int GetColor();

int GetType();

int GetCoord();

int GetNum();

//изменяют поле на противоположное значение

void ChangeColor();

void ChangeType();

//дружественные классы

friend class ListOfCheckers;//список шашек

friend class LChIterator;//итератор для перемещения по списку шашек

protected://методы, предназначенные для использования классом ListOfCheckers

void SetNextNum(int num);

void SetPrevNum(int num);

int GetNextNum();

int GetPrevNum();

};

1. Класс ListOfCheckers и класс LChIterator

Класс *ListOfCheckers* является типом, представляющим список шашек на массиве. Необходимость его создания обусловлена тем, что в программе необходимо как обращаться к шашке по индексу, так и часто удалять и добавлять шашки. Класс *LChIterator* является итератором к этому классу.

class ListOfCheckers //список шашек

{

Checker List[13]; //12 шашек + голова, чтобы список никогда не был пустым

//шашки нумеруются от 1 до 12

public:

ListOfCheckers();

~ListOfCheckers() {}

//генерирует первоначальную позицию по заданным массивам типов и координат

void GenerateInitialPosition(int color, int\* types, int\* coords, int n);

void Insert(int num); //вставляет шашку в список

void Delete(int num); //удаляет шашку из списка

int IsEmpty();//проверяет, пустой ли список

Checker& operator[] (int i); //оператор индексации

void Clean(); //очищает стек

void Bind(int n);//связывает список друг за другом

//итератор

typedef LChIterator iterator;

iterator begin() { return iterator(&(List[List[0].GetNextNum()]),List); }

iterator end() { return iterator(List, List); }

};

class LChIterator //итератор

{

Checker\* begin; //указатель на голову списка

Checker\* p; //шашка, на которую указывает итератор

public:

LChIterator() {}

//инициализатор полей

LChIterator(Checker\* p\_, Checker\* begin\_) :p(p\_), begin(begin\_) {};

//арифметические операторы

LChIterator& operator++();

LChIterator& operator--();

LChIterator& operator++(int);

LChIterator& operator--(int);

//операторы приведения типов

operator Checker\* ();

operator int();

//операторы разыменования

Checker\* operator->();

Checker operator\* (LChIterator it);

//операторы сравнения

friend int operator==(LChIterator it, Checker\* pch);

friend int operator!=(LChIterator it1, LChIterator it2);

};

1. Класс Board

Доска представляет собой одномерный массив из указателей на Checker. Это необходимо для того, чтобы, зная координату, сразу находить соответвствующую ей шашку.

class Board

{

Checker\* board[64];

public:

Board();

Board(ListOfCheckers& white, ListOfCheckers& black);//заполняет доску фигурами

void Set(ListOfCheckers& white, ListOfCheckers& black); //заполняет доску фигурами

int IsEmpty(int cell);//проверяет, пустая ли клетка

Checker\*& operator [](int cell);//индексация

void Clean();//очищает доску

};

1. Класс Move

При реализации алгоритмов выясняется, что нужно как-то хранить ход шашки. Следующий класс предоставляет все необходимые для этого поля и методы.

class Move

{

int Color;//цвет шашки, которая ходит

int StartCoord;//начальные координаты хода

int FinalCoord;//конечные координаты хода

int Num;//идентификатор перемещающейся шашки

int Type;//изменился ли тип шашки во время хода

int NEaten;//число съеденных шашек

int Eaten[12] = { 0 };//массив съеденных шашек

public:

Move() :Color(0), Type(0), StartCoord(0), FinalCoord(0), Num(0), NEaten(0) {}

//операторы сравнения

friend int operator==(const Move& m1, const Move& m2);

friend int operator!=(const Move& m1, const Move& m2);

~Move() {}

//устанавливают поля класса

void Set(int \_color, int \_s\_coord, int \_f\_coord, int \_num, int \_type);

void SetColor(int color);

void SetStartCoord(int coord);

void SetFinalCoord(int coord);

void SetNum(int num);

void SetType(int type);

void SetNEaten(int neaten);

void SetEaten(int\* arrEaten);//записывает номера съеденных шашек в массив

//возвращают значения полей класса

int GetColor();

int GetStartCoord();

int GetFinalCoord();

int GetNum();

int GetType();

int GetNEaten();

void GetEaten(int\* arrEaten);//возвращает съеденные шашки в массив arrEaten

};

1. Класс Cache

На каждом шаге рекурсии необходимо где-то хранить возможные ходы. Можно было бы создавать на каждом шаге рекурсии массив ходов, но, во-первых, это может занимать некоторое время, во-вторых, выделять место пришлось бы с большим запасом, поскольку неизвестно, сколько ходов может быть сгенерировано для определенной позиции. Поэтому мы выделяем место для хранения ходов в виде большого массива один раз в самом начале, а во время работы рекурсии просто записываем туда ходы. Очищение происходит посредством перемещения указателя на конец массива в сохраненную точку.

Такая структура хранения представлена следующим классом.

class Cache

{

private:

int Size;//размер кэша

Move \*pData;//указатель на начало

Move \*pLast;//указатель на конец

public:

Cache():Size(0), pData(0), pLast(0){};

Cache(int size) : Size(size);//выделяет динамически массив размера size

void Push(Move data); //запись данных в кэш

Move& Pop();//извлечение из кэша

Move\* GetpLast();//получить указатель на конец кэша

void Rollback(Move \*saved);//очищение кэша до сохраненной позиции saved

int CurPos(); //получить текущий размер кэша

Move operator[](int i);//индексирование

bool IsFull();//проверяет, заполнен ли кэш

bool IsEmpty(); //проверяет, пуст ли кэш

void Clean(); //очищает кэш полностью

~Cache();

};

1. См п.2 [↑](#footnote-ref-2)