МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Проект разработка игры «Борьба за жизнь»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к выпускной работе

ОБУЧАЮЩИЙСЯ:

Москалев Юрий Антонович \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2022

**Задание 10: игра «Борьба за жизнь. Описание задания.**

Два игрока имеют по n шашек и играют в поле m на m клеток, делая ходы по очереди.

Возможные ходы:

1) переставить свою шашку на соседние свободные клетки

2) переставить свою шашку через поле, если оно занято другой шашкой.

Если шашка с трѐх сторон окружена шашками противника, она снимается с поля. Если

пустая клетка с 3-х сторон окружена шашками одного игрока, на еѐ место ставится новая

шашка этого игрока. Цель игры – убрать с доски все шашки противника.

**Дополнение к правилам игры**

В игре присутствуют игровые ситуации, которые приводят к завершению игры, не выполняя главную цель из условия задачи. В определенный момент всё игровое поле заполняется клетками двух игроков и совершить следующий ход становится невозможно. Так же существует ситуация, когда у одного игрока остаётся одна или несколько клеток, которые не могут сделать ход, однако карта может быть еще свободна для заполнения.

Для решения этой проблемы главная цель игры была изменена. Теперь победителем является тот, у кого больше его клеток его команды и противник не может сделать ни одного доступного хода. После этого игра завершается.

**Начало игры**

После запуска приложения игра встречает нас окном с именем игры

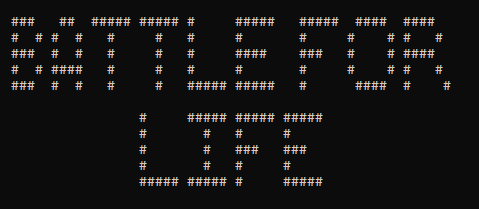


Рис. 1 Картинка стартового окна

Далее нас встречает главное меню, где можно выбрать одно из предложенных действий.

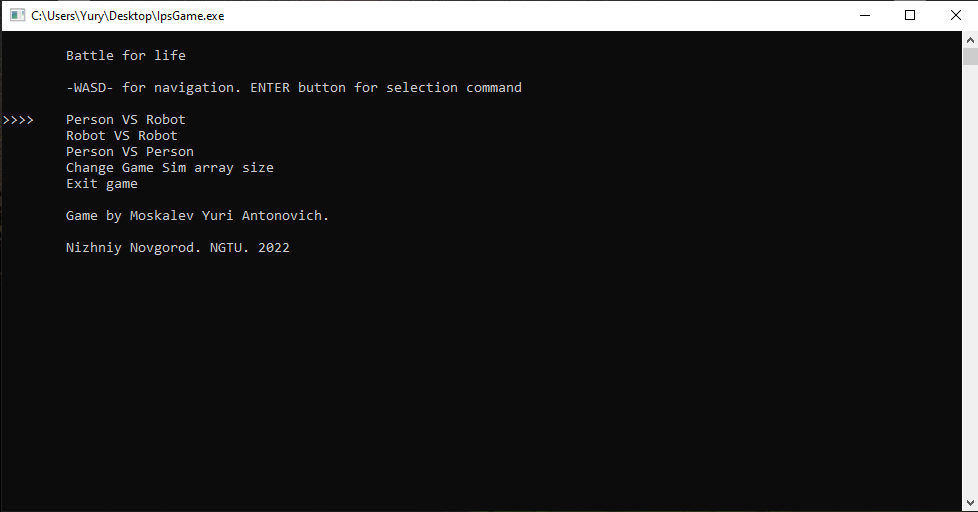


Рис. 2 Главное меню игры

Здесь можно выбрать несколько вариантов пользователей игры. Можно устроить игру один на один с ботом, можно устроить бой искусственного интеллекта против другого, можно устроить битву двух игроков на одном поле. Так же можно выбрать размер поля от 3 на 3 до   
7 на 7 на границах поля, при этом высота и ширина может отличаться друг от друга. Такие ограничения были сделаны в виду особенностей игры. Если делать поле меньше 3 на 3 включительно, то игровые правила будут не работать логически. Если сделать поле больше, то оно не будет умещаться в размер экрана консоли.

Для ускорения игры выберем размер поля 5 на 3.

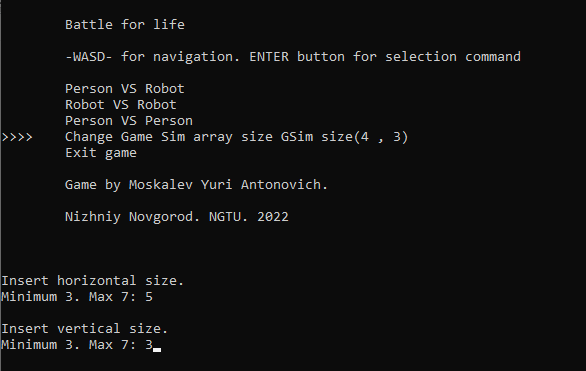


Рис. 3 Меню после выбора размера поля

После чего мы выбираем игру с ботом. После чего мы наблюдаем следующую картину:

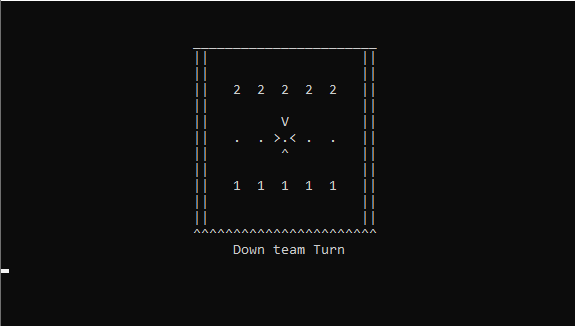


Рис. 4 Игровое поле.

**Игровой процесс**

Здесь мы можем наблюдать игровое поле. На нем располагаются клетки с числами и точками. Число – это номер команды, которой принадлежит эта клетка. Цифра 2 – это верхняя команда, второй игрок, цифра 1 – это нижняя команда, первый игрок. Игра начинается с хода человека. Сторону текущего игрока показывают стрелки сверху или снизу. Так же в центре поля появляется выделенная стрелками точка – это курсор, которым можно управлять на кнопки перемещения.

Для того, чтобы подвинуть нашу шашку, нужно подвинуть курсор на нужную нам клетку и затем нажать ENTER для её выбора. Далее мы наблюдаем следующее:

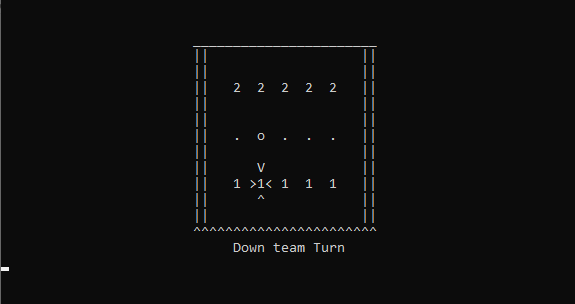
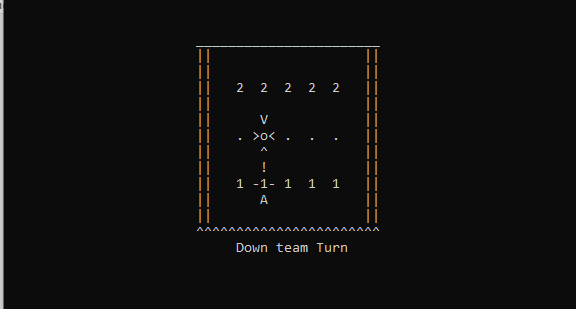


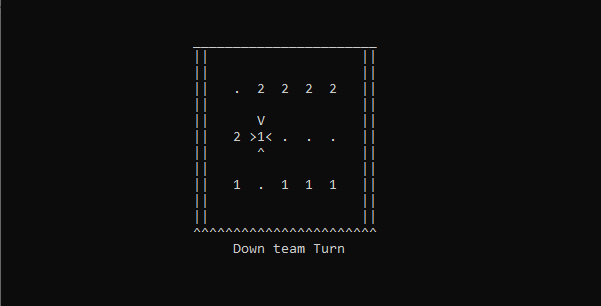
Рис. 5 Событие выбора клетки

После нажатия ENTER мы видим кружок недалеко от своей клетки. Это доступное для перемещения поле. Теперь мы всё так же можем двигать свой курсор в любую сторону. На месте выбранной клетки остаётся курсор немного другого вида, чтобы указывать на выбранную шашку.

  
Рис. 6 Событие перемещения выбранной клетки

Затем мы можем нажать ENTER, и наша клетка переместится на доступное место. Если мы передумаем, то можно нажать BACKSPACE и тогда выбор отменится. Если нажать на недоступную для перемещения клетку, то курсор сбросится и вернется в предыдущую позицию.

Далее следует ход компьютера. Оно сопровождается изменением позиции курсора (это курсор другого игрока) и всплывающим сверху табличкой «ход верхней команды». После чего бот двигает клетку, которую «посчитал нужным подвинуть». Об устройстве ИИ будет рассказано после раздела с игровым процессом.

  
Рис. 7 Новый ход нижнего игрока

Под нашим курсором мы можем обнаружить пространство между тремя нашими клетками. По условиям игры, мы можем создать там свою шашку. Двигаем курсор и дважды нажимаем ENTER. После этого на нашем месте появится шашка из нашей команды, после чего ход переходит другому игроку.

Далее происходит интересная ситуация. Бот «видит», что он может окружить нашу клетку, чем успешно пользуется, чем мгновенно убирает нашу шашку с поля

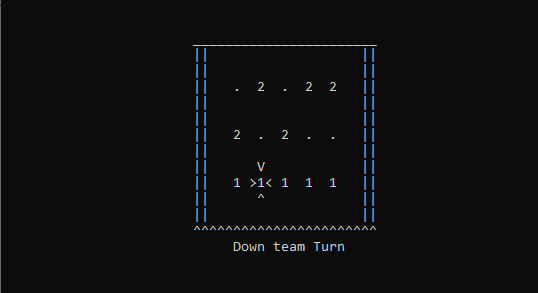


Рис. 8 Создание и уничтожение шашек

Сейчас мы видим, что у бота появилось два поля для создания шашек. В свою очередь, по правилам игры, мы можем перемесить нашу шашку через другую, чем можем попасть в левый верхний угол.

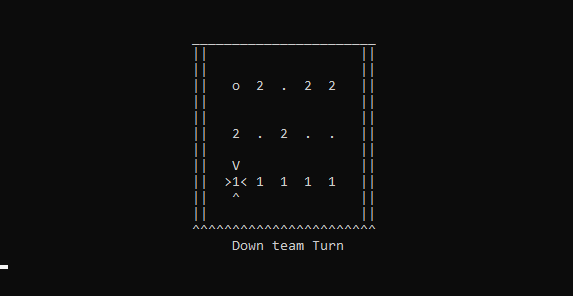
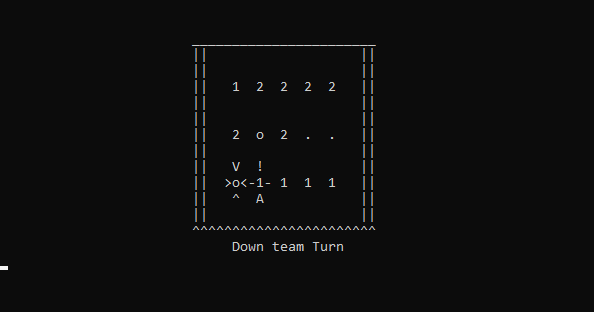


Рис. 9 Доступные для перемещения позиции

Перемещаем клетку и начинается ход компьютера. Далее мы видим, что бот создал свою шашку на свободном месте. Далее мы двигаем свою шашку на позицию, которая у нас открылась после перемещения. Мы можем наблюдать, что слева и сверху есть доступные позиции.

  
Рис. 10 Текущая игровая ситуация

Далее происходит несколько шагов от меня и от бота. Ситуация на поле была не в лучшую сторону в связи с потерей своей шашки. Бот старается выслеживать шашки, которые он может устранить, старается создавать новые и убирать свои шашки под угрозой.

В определенный момент количество моих шашек резко уменьшилось, как и количество ходов. Далее возникает ситуация, когда приходится делать в ходы в явно невыгодное положение.

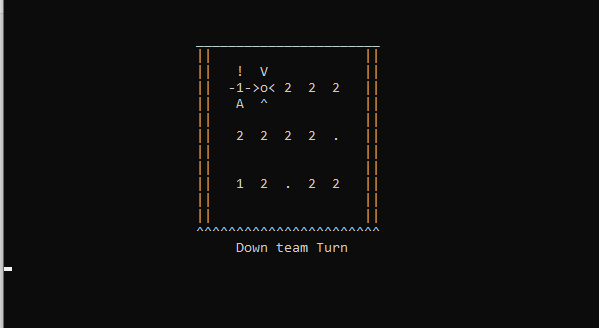


Рис. 11 Безвыходный проигрышный ход.

После этого, у меня появляется два возможных хода, однако оба проигрышные. После моего последнего хода бот забирает последнюю шашку.

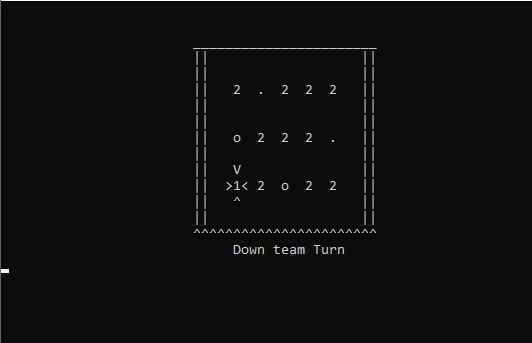
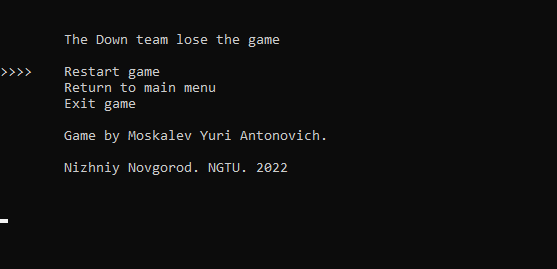


Рис. 12 Последний ход игрока.

После этого игра завершается и нас перекидывает на меню конца игры, где можно либо перезапустить игру, либо вернуться в главное меню, либо покинуть игру.

  
Рис. 13 Меню окончания игры.

Во врем игры в любой момент времени можно было нажать на кнопку ESC и вызвать меню паузы, где так же можно было вернуться в главное меню и закончить игру или продолжить игру.

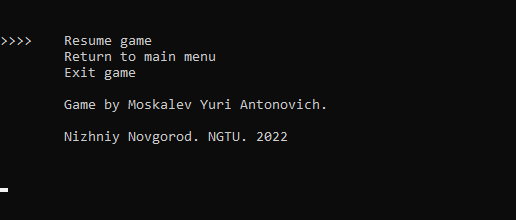


Рис. 14 Меню паузы игры.

**Описание программного кода**

Общая структура приложения

Игра разделена на несколько крупных модулей, которые входят в состав главного модуля GameCore. Однако были предприняты решения по уменьшению зависимости внутренних модулей друг с другом. Для этого был создан класс «OEDataBase» - База данных, который хранит в себе крупные данные или общие данные. Так все ресурсы игры будут находиться в одном месте и при этом быть доступны через интерфейс базы данных. Для доступа к базе данных был создан класс «OEDBConnector» - коннектор, который будет входить в состав других модулей, либо же быть базовым классом для уникальных коннекторов, при этом располагая основным способом доступа к базе данных – указатель на неё.

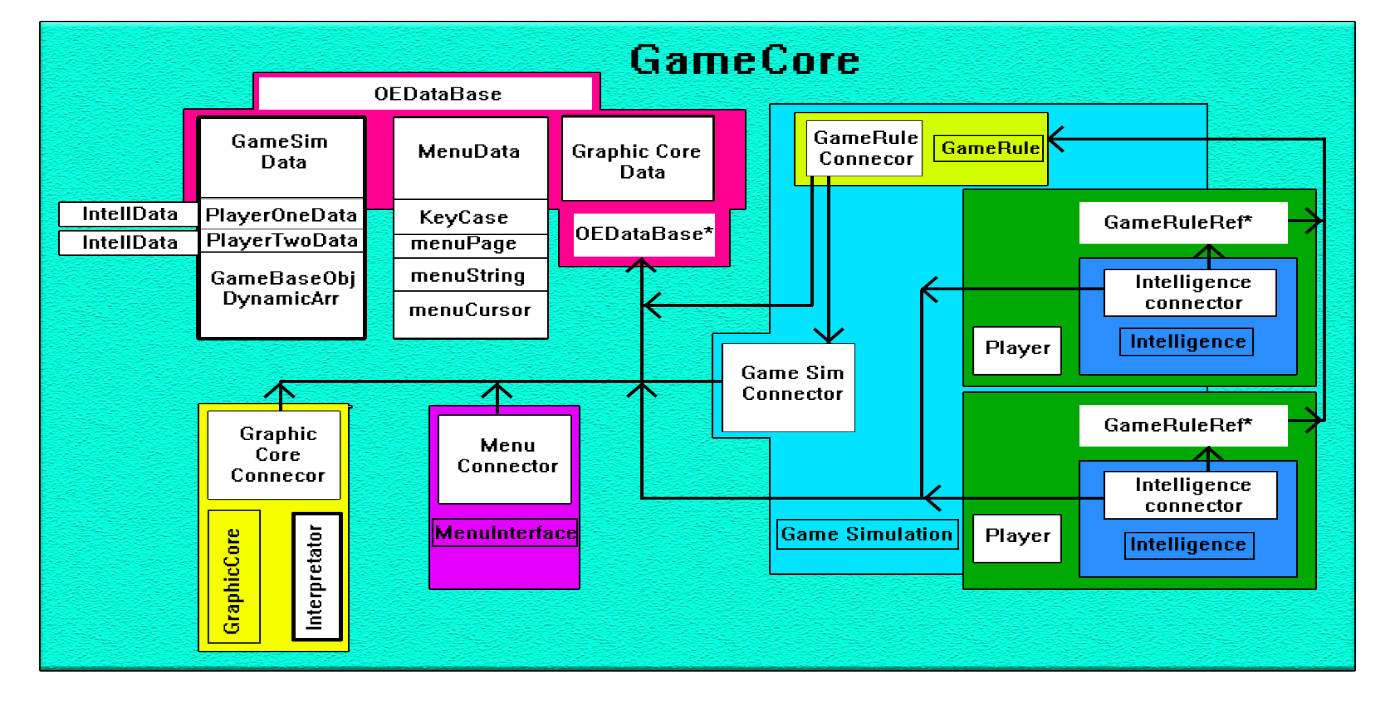


Рис. 15 Общая структура зависимостей модулей

Использование базы данных и коннекторов позволяет удобно считывать и записывать данные для общего пользования, при этом имея возможность создавать дополнительную логику при выдаче и записи данных.

* Модуль GraphicCore – отрисовывает на экране текущую графическую информацию
* Модуль Interpretator – переводит данные из модуля игровой симуляции в данные, поддерживаемые графическим ядром.
* Модуль Menu – обрабатывает ввод с клавиатуры, перенаправляет на необходимые страницы меню, задаёт модулям их режим работы и перенаправляет сигналы с клавиатуры в модули.
* Модуль GameSimulation – ядро симуляции игрового процесса. Содержит правила игры, двух игроков и варианты обработки сигналов с клавиатуры. Так же каждый такт цикла обновляет информацию об игровом поле.
* Модуль Player – это модуль участника игры. Выполняет команды, приходящие из модуля Intelligence
* Модуль Intelligence – это базовый модуль, который выбирает какие действия сделать на данный ход.   
  PersonIntelligence – это непосредственно обработчик команд с клавиатуры от живого игрока.   
  ArtificialIntelligence – это модуль Искусственного интеллекта. Компьютер самостоятельно выбирает какой сделать ход.
* Модуль GameRules – это модуль, содержащий правила и логику игры.

**Устройство искусственного интеллекта**

ИИ на свой ход находит все доступные ходы среди всех своих клеток. Далее ИИ проверяет каждый возможный ход и даёт оценку произошедшего.

* При изменении количества клеток на поле, меняется оценка.
* При наличии возможности убрать шашку противника, выдается более высокая оценка хода
* При возможности уничтожения своей клетки, оценка хода уменьшается и бот отступает
* Если около своих клеток имеется возможность создаёт свою клетку, выдается высокая оценка этого хода.

Далее вычисляется наибольшая оценка хода, которой соответствует один из множества шагов и ИИ делает передает команду на исполнение классу игрока.

При наличии одинаковых оценок ходов, во избежание бесконечного повторения хода, выбираются максимальные одинаковые оценки ходов и среди них случайным образом выбирается ход. В любом случае действие с высокой оценкой более важно и бот его выполняет. Начальные ходы ИИ имеет всегда одинаковые значения, поэтому изначально случайно перемещается по карте, пока не начнет создавать свои шашки и не найдет возможности окружить шашки противника.

Из-за того, что имеется случайность в действиях у ИИ, игра двух ботов каждый раз уникальна, при этом, оба ИИ не знают действий друг друга, из-за чего игра становится непредсказуемой.

**Логика работы приложения**

1. Запуск приложения
2. Базовая инициализация всех модулей. Передача указателей между модулями.
3. Выдача на экран сообщения об имени игры
4. Запуск главного цикла игры
5. Ожидание сигнала с клавиатуры
6. Обработка сигнала, при наличии.
7. Изменение состояния страниц меню по результатам обработки
8. Запуск обработчика команд симуляции игры
9. При команде, запускающая игровую симуляцию, выполняется один такт симуляции. Такт симуляции не равен одному ходу игрока. Ход игрока заканчивается в зависимости от команды с клавиатуры или действию ИИ
10. Запуск обработчика команд графического ядра.
11. Отрисовка на экране текущей страницы.
12. Проверка на окончание работы приложения
13. Повторение действий с пункта 5

**Программный код**

Среда разработки: Microsoft Visual studio 2019.

Используемые библиотеки: Стандартная библиотека STD, стандартная библиотека Windows.h

**Заголовочные файлы:**

* AIRecursionData.h – объявление функций и классов доп. Информации для Искусственного интеллекта
* ArtificialIntelligence.h – объявление функций и классов ИИ.
* BattleForLifeGameComponents.h - объявление функций и классов игровых правил
* GameBaseComponents.h – дополнительные функции и классы для игры
* GameCore.h – объявление класса главного игрового ядра
* GameSimConnector.h – класс «Коннектор» для нескольких модулей
* GameSimulation.h – класс ядра игровой симуляции
* GraphicCore.h – класс графического ядра
* GrCoreGameSimDataInterpretator.h – класс интерпретатора информации из игровой симуляции
* Intelligence.h – описание базового класса «интеллекта игрока»
* Menu.h – класс «меню игры»
* OeDataBase.h – класс базы данных
* OEDataBaseData.h – класс дополнительных сгруппированных данных из модулей
* PersonIntelligence.h – объявление функций и классов действий живого игрока
* Player.h – класс виртуального игрока

**Дополнительные заголовки:**

* Vector2DTemplate.h – шаблон динамического многомерного вектора
* GameSimStructs.h – дополнительные данные от игровой симуляции
* GraphicCoreStructs.h – дополнительные структуры для граф. Ядра

**Файлы реализации:**

* AIRecursionData.cpp – функции доп. Обработки ходов ИИ
* ArtificialIntelligence.cpp – функции главной логики ИИ.
* BattleForLifeGameComponents.cpp - функции правил игры
* GameBaseComponents.cpp – функции дополнительных элементов игры
* GameCore.cpp –реализация главной логики всего приложения
* GameSimulation.cpp – реализация логики работы ядра игровой симуляции
* GraphicCore.cpp – реализация логики работы граф. ядра
* GrCoreGameSimDataInterpretator.cpp – функции интерпретатора
* Menu.cpp – содержит реализацию страниц меню и их логики
* PersonIntelligence.cpp – содержит обработчик команд с клавиатуры живого игрока
* Player.cpp – реализация логики поведения виртуального игрока
* IpsGame.cpp – расположение функции **main()**

Текст файлов

**AIRecursionData.cpp**  
#include "AIRecursionData.h"

bool isPlaceAcceptableToMove(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* curSituation, int x, int y);

void fractionCheck(\_FRACTION\_INT\_ curFrac, unsigned short int\* counter1, unsigned short int\* counter2)

{

switch (curFrac)

{

case \_BFL\_TEAM\_UP\_SIDE:

{

(\*counter1)++;

break;

}

case \_BFL\_TEAM\_DOWN\_SIDE:

{

(\*counter2)++;

break;

}

default:

break;

}

}

\_FRACTION\_INT\_ AIRecursionData::getMyEnemy(\_FRACTION\_INT\_ me)

{

switch (me)

{

case \_BFL\_TEAM\_UP\_SIDE:

{

return \_BFL\_TEAM\_DOWN\_SIDE;

break;

}

case \_BFL\_TEAM\_DOWN\_SIDE:

{

return \_BFL\_TEAM\_UP\_SIDE;

break;

}

case \_BFL\_TEAM\_NONE:

{

return \_BFL\_TEAM\_NONE;

break;

}

default:

break;

}

return \_BFL\_TEAM\_NONE;

}

AISquareSurroundsInfo whatFractionSurrounds(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* currSituation, int x, int y)

//unsigned short MODE = \_AI\_INTEL\_DATA\_MODE\_SHOW\_FRACTION)

{

AISquareSurroundsInfo retVal;

int fractionNone = 0;

unsigned short int contUpTeam = 0, countDownTeam = 0;

int left = 0, up = 0, right = 0, down = 0;

if (currSituation->isCoordInLimits( x - 1, y))

{

left = \*currSituation->getElem(x - 1, y);

}

if (currSituation->isCoordInLimits(x + 1, y))

{

right = \*currSituation->getElem(x + 1, y);

}

if (currSituation->isCoordInLimits(x, y - 1))

{

up = \*currSituation->getElem(x, y - 1);

}

if (currSituation->isCoordInLimits(x, y + 1))

{

down = \*currSituation->getElem(x, y + 1);

}

fractionCheck(up, &contUpTeam, &countDownTeam);

fractionCheck(right, &contUpTeam, &countDownTeam);

fractionCheck(down, &contUpTeam, &countDownTeam);

fractionCheck(left, &contUpTeam, &countDownTeam);

retVal.downTeamCount = countDownTeam;

retVal.upTeamCount = contUpTeam;

if (retVal.upTeamCount >= 3 )

{

retVal.cellFraction = \_BFL\_TEAM\_UP\_SIDE;

}

else if(retVal.downTeamCount >= 3)

{

retVal.cellFraction = \_BFL\_TEAM\_DOWN\_SIDE;

}

else

{

retVal.cellFraction = \_BFL\_TEAM\_NONE;

}

return retVal;

}

bool AIRecursionData::isEnemySurroundedByMe(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* currSituation, \_FRACTION\_INT\_ myFract, int x, int y)

{

\_FRACTION\_INT\_ chosenCell;

chosenCell = \*currSituation->getElem(x, y);

if (chosenCell == getMyEnemy(myFract) && whatFractionSurrounds(currSituation, x, y).cellFraction == myFract)

{

return true;

}

return false;

}

bool canCreateCellInSurrounding(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* currSituation, \_FRACTION\_INT\_ myFract, int x, int y)

{

if (whatFractionSurrounds(currSituation, x, y).cellFraction == myFract)

{

return true;

}

else

{

return false;

}

}

std::vector<AIVecPosFromPosTo> AIRecursionData::showPlacesToCreateMyCquare(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* curSituation,

\_FRACTION\_INT\_ fract, std::vector<AIVecPosFromPosTo> myTeamPositions)

{

int maxX = curSituation->getSizeHorizonal();

int maxY = curSituation->getSizeVertical();

std::vector<AIVecPosFromPosTo> retVec;

std::vector<AIVecPosFromPosTo>::iterator iter = myTeamPositions.begin();

while (iter != myTeamPositions.end())

{

AIVecPosFromPosTo it = \*iter;

if (whatFractionSurrounds(curSituation, it.posTo.getX(), it.posTo.getY()).cellFraction == fract)

{

it.action = \_BFL\_INTELLIGENCE\_CREATE\_CQUARE;

retVec.push\_back(it);

}

++iter;

}

return retVec;

}

bool isPlaceAcceptableToMove(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* curSituation, int x, int y)

{

if (!curSituation->isCoordInLimits(x, y))

{

return false;

}

if (\*curSituation->getElem(x, y) == \_BFL\_TEAM\_NONE)

{

return true;

}

return false;;

}

int AIRecursionData::createMyTeamSquareValue(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* currSiation, \_FRACTION\_INT\_ myFraction)

{

int maxX = currSiation->getSizeHorizonal();

int maxY = currSiation->getSizeVertical();

std::vector<AIVecPosFromPosTo> canCreateSquVec;

canCreateSquVec = showPlacesToCreateMyCquare(currSiation, myFraction,

getAllPossibleToMoveMyCells(currSiation, myFraction));

return canCreateSquVec.size() \* \_AI\_RECURSION\_CAN\_CREATE\_MY\_SQUARE;

}

int AIRecursionData::killEnemySquareValue(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* currSiation, \_FRACTION\_INT\_ myFraction)

{

int maxX = currSiation->getSizeHorizonal();

int maxY = currSiation->getSizeVertical();

int enemyWictimCount = 0;

std::vector<AIVecPosFromPosTo> mySquares;

mySquares = getAllPossibleToMoveMyCells(currSiation, myFraction);

std::vector<AIVecPosFromPosTo>::iterator iter = mySquares.begin();

while (iter != mySquares.end())

{

AIVecPosFromPosTo it = \*iter;

AIRecursionData innerRec;

innerRec = \*this;

innerRec.moveSquare(it);

std::vector<vecI2d> allSquaresOfMyTeam;

for (int y = 0; y < innerRec.curSituation.getSizeVertical(); y++)

{

for (int x = 0; x < innerRec.curSituation.getSizeHorizonal(); x++)

{

if (isSquareInMyTeamRecurce(&(innerRec.curSituation), myFraction, x, y))

{

allSquaresOfMyTeam.push\_back(vecI2d(x, y));

}

}

}

std::vector<vecI2d>::iterator innerIter = allSquaresOfMyTeam.begin();

while (innerIter != allSquaresOfMyTeam.end())

{

vecI2d curSquare;

curSquare = \*innerIter;

int curX = curSquare.getX(), curY = curSquare.getY();

int up = curY - 1, down = curY + 1,

left = curX - 1, right = curX + 1;

if (currSiation->isCoordInLimits(curX, up))

{

if (isEnemySurroundedByMe(currSiation, myFraction , curX, up))

{

enemyWictimCount++;

}

}

if (currSiation->isCoordInLimits(curX, down))

{

if (isEnemySurroundedByMe(currSiation, myFraction, curX, down))

{

enemyWictimCount++;

}

}

if (currSiation->isCoordInLimits(left, curY))

{

if (isEnemySurroundedByMe(currSiation, myFraction, left, curY))

{

enemyWictimCount++;

}

}

if (currSiation->isCoordInLimits(right, curY))

{

if (isEnemySurroundedByMe(currSiation, myFraction, right, curY))

{

enemyWictimCount++;

}

}

++innerIter;

}

++iter;

}

return enemyWictimCount \* \_Ai\_RECURSION\_CAN\_KILL\_ENEMY;

}

#define \_NEIGHBOUR\_NEAR\_POSITION 1

std::vector<vecI2d> showAllPosiblePositions(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* curSituation,

\_FRACTION\_INT\_ frac, int x, int y)

{

std::vector<vecI2d> retVal;

if (!curSituation->isCoordInLimits(x, y))

{

vecI2d tempVec;

retVal.push\_back(tempVec);

return retVal;

}

int

leftCell = x - 1, leftLeftCell = leftCell - 1,

rightCell = x + 1, rightRightCell = rightCell + 1,

upCell = y - 1, upUpCell = upCell - 1,

downCell = y + 1, downDownCell = downCell + 1;

if (!isPlaceAcceptableToMove(curSituation, leftCell, y))

{

if (isPlaceAcceptableToMove(curSituation, leftLeftCell, y))

{

vecI2d tempVec;

tempVec = vecI2d(leftLeftCell, y);

retVal.push\_back(tempVec);

}

}

else

{

vecI2d tempVec;

tempVec= vecI2d(leftCell, y);

retVal.push\_back(tempVec);

}

if (!isPlaceAcceptableToMove(curSituation, rightCell, y))

{

if (isPlaceAcceptableToMove(curSituation, rightRightCell, y))

{

vecI2d tempVec;

tempVec = vecI2d(rightRightCell, y);

retVal.push\_back(tempVec);

}

}

else

{

vecI2d tempVec;

tempVec = vecI2d(rightCell, y);

retVal.push\_back(tempVec);

}

/////////////

if (!isPlaceAcceptableToMove(curSituation, x, upCell))

{

if (isPlaceAcceptableToMove(curSituation, x, upUpCell))

{

vecI2d tempVec;

tempVec = vecI2d(x, upUpCell);

retVal.push\_back(tempVec);

}

}

else

{

vecI2d tempVec;

tempVec= vecI2d(x, upCell);

retVal.push\_back(tempVec);

}

if (!isPlaceAcceptableToMove(curSituation, x, downCell))

{

if (isPlaceAcceptableToMove(curSituation, x, downDownCell))

{

vecI2d tempVec;

tempVec = vecI2d(x, downDownCell);

retVal.push\_back(tempVec);

}

}

else

{

vecI2d tempVec;

tempVec = vecI2d(x, downCell);

retVal.push\_back(tempVec);

}

return retVal;

}

#define shortLimitCheck curSituation->isCoordInLimits

#define NOT !

#define \_AI\_NOT\_EQUAL\_TEAM\_NONE != \_BFL\_TEAM\_NONE

int AIRecursionData::calcSquareVal(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* curSituation, int X, int Y, \_FRACTION\_INT\_ frac)

{

unsigned short int enemyCount = 0;

unsigned short int myTeamCount = 0;

\_FRACTION\_INT\_ tempTeam;

for (int y = 0; y < curSituation->getSizeVertical(); y++)

{

for (int x = 0; x < curSituation->getSizeHorizonal(); x++)

{

if (shortLimitCheck(x, y) && curSituation->getElem(x, y) \_AI\_NOT\_EQUAL\_TEAM\_NONE)

{

tempTeam = \*curSituation->getElem(x, y);

if (frac == tempTeam)

{

myTeamCount++;

}

if (tempTeam == getMyEnemy(frac))

{

enemyCount++;

}

}

}

}

int retSquareValue = 0;

retSquareValue = \_AI\_MY\_TEAM\_SQUARES\_VALUE \* myTeamCount - enemyCount \* \_AI\_ENEMY\_TEAM\_SQUARES\_VALUE;

int canKillEnemysCount = AIRecursionData::killEnemySquareValue(curSituation, frac);

retSquareValue += canKillEnemysCount;

int canBeKilledByEnemysCount = - AIRecursionData::killEnemySquareValue(curSituation, getMyEnemy(frac));

retSquareValue += canBeKilledByEnemysCount;

return retSquareValue;

}

#undef shortLimitCheck

#undef NOT

#undef \_AI\_NOT\_EQUAL\_TEAM\_NONE

bool AIRecursionData::isSquareInMyTeamRecurce(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* curSituation, \_FRACTION\_INT\_ curFraction, int x, int y)

{

\_FRACTION\_INT\_ curObject;

curObject = \*curSituation->getElem(x, y);

if (curObject != curFraction)

{

return false;

}

return true;

}

void AIRecursionData::moveSquare(AIVecPosFromPosTo newPos)

{

if (!curSituation.isCoordInLimits(newPos.posTo.getX(), newPos.posTo.getY()))

{

return;

}

\_FRACTION\_INT\_ tempElem;

tempElem = \*curSituation.getElem(newPos.posFrom.getX(), newPos.posFrom.getY());

curSituation.setElem(\_BFL\_TEAM\_NONE, newPos.posFrom.getX(), newPos.posFrom.getY());

curSituation.setElem(tempElem, newPos.posTo.getX(), newPos.posTo.getY());

}

std::vector<AIVecPosFromPosTo> AIRecursionData::getAllPossibleToMoveMyCells(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* currSituation, \_FRACTION\_INT\_ myTeam)

{

std::vector<AIVecPosFromPosTo> retVec;

std::vector<vecI2d> allSquaresOfMyTeam;

for (int y = 0; y < currSituation->getSizeVertical(); y++)

{

for (int x = 0; x < currSituation->getSizeHorizonal(); x++)

{

if (isSquareInMyTeamRecurce(currSituation, myTeam, x, y))

{

allSquaresOfMyTeam.push\_back(vecI2d(x, y));

}

}

}

std::vector<vecI2d>::iterator myTeamSqIter = allSquaresOfMyTeam.begin();

while (myTeamSqIter != allSquaresOfMyTeam.end())

{

vecI2d curSq;

curSq = (\*myTeamSqIter);

std::vector<vecI2d> posibbleCurTurns = showAllPosiblePositions(currSituation, myTeam,

curSq.getX(), curSq.getY());

if (!posibbleCurTurns.empty())

{

std::vector<vecI2d>::iterator possibleTurnIter = posibbleCurTurns.begin();

while (possibleTurnIter != posibbleCurTurns.end())

{

vecI2d curPossibleTurn = \*possibleTurnIter;

if (isPlaceAcceptableToMove(currSituation, curPossibleTurn.getX(), curPossibleTurn.getY()))

{

AIVecPosFromPosTo FTPosToRetValVec;

FTPosToRetValVec.posFrom = curSq;

FTPosToRetValVec.posTo = curPossibleTurn;

FTPosToRetValVec.value = \_BFL\_INTELLIGENCE\_MOVE\_CELL\_CUSTOM;

retVec.push\_back(FTPosToRetValVec);

}

++possibleTurnIter;

}

}

++myTeamSqIter;

}

return retVec;

}

std::vector<AIVecPosFromPosTo> AIRecursionData::getAllPossibleTurns(\_FRACTION\_INT\_ myTeam)

{

std::vector<AIVecPosFromPosTo> retVec;

retVec = getAllPossibleToMoveMyCells(&curSituation, myTeam);

std::vector<AIVecPosFromPosTo> createCellTurns;

createCellTurns = showPlacesToCreateMyCquare(&curSituation, myTeam, retVec);

if (!createCellTurns.empty())

{

std::vector<AIVecPosFromPosTo>::iterator iter = createCellTurns.begin();

while (iter != createCellTurns.end())

{

retVec.push\_back(\*iter);

++iter;

}

}

return retVec;

}

bool AIRecursionData::wasPosInTurnList(vecI2d curPos)

{

for (int i = 0; i < previousPos.size(); i++)

{

if (curPos == previousPos[i])

{

return true;

}

}

return false;

}

#undef brk

**AIRecursionData.h**

#ifndef \_AI\_RECURSION\_DATA\_H

#define \_AI\_RECURSION\_DATA\_H

#include "Vector2DTemplate.h"

#include "BattleForLifeGameComponents.h"

#include "GameSimStructs.h"

#include "Intelligence.h"

#include <list>

#include <math.h>

#include <limits>

#define \_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR DynamicIntArray2DTemplated<\_FRACTION\_INT\_>

#define \_AI\_INTELLIGENCE\_ERR\_CURRENT\_FRACTION\_NOT\_SET "AI\_INTELLIGENCE\_ERR\_CURRENT\_FRACTION\_NOT\_SET"

#define \_AI\_INTEL\_DATA\_MODE\_SHOW\_FRACTION 1

#define \_AI\_INTEL\_DATA\_MODE\_SHOW\_MAX\_COUNT 2

class AIVecPosFromPosTo

{

public:

AIVecPosFromPosTo(){}

AIVecPosFromPosTo(vecI2d newPosFrom, vecI2d newPosTo):

posFrom(newPosFrom) , posTo(newPosTo){}

vecI2d posFrom = \_BFL\_RULES\_POSSIBLE\_POS\_NOT\_FOUND\_VEC;

vecI2d posTo = \_BFL\_RULES\_POSSIBLE\_POS\_NOT\_FOUND\_VEC;

int value = 0;

int action = \_BFL\_INTELLIGENCE\_NO\_ACTION;

bool operator>(AIVecPosFromPosTo comp)

{

if (value >= comp.value)

{

return true;

}

else

{

return false;

}

}

bool operator<(AIVecPosFromPosTo comp)

{

if (value <= comp.value)

{

return true;

}

else

{

return false;

}

}

bool operator==(AIVecPosFromPosTo comp)

{

if (value == comp.value)

{

return true;

}

else

{

return false;

}

}

};

class AISquareSurroundsInfo

{

public:

\_FRACTION\_INT\_ cellFraction = \_BFL\_TEAM\_NONE;

unsigned short upTeamCount = 0;

unsigned short downTeamCount = 0;

};

class AIRecursionData

{

private:

std::vector<vecI2d> previousPos;

std::list<AIVecPosFromPosTo> prevPosList;

public:

AIRecursionData() {}

AIRecursionData(\_FRACTION\_INT\_ fraction) :

AiFraction(fraction){}

\_FRACTION\_INT\_ getMyEnemy(\_FRACTION\_INT\_ me);

bool wasPosInTurnList(vecI2d curPos);

bool isSquareInMyTeamRecurce(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* curSituation, \_FRACTION\_INT\_ curFraction, int x, int y);

DynamicIntArray2DTemplated<\_FRACTION\_INT\_> curSituation;

std::vector<AIVecPosFromPosTo> getAllPossibleTurns(\_FRACTION\_INT\_ myTeam);

\_FRACTION\_INT\_ AiFraction;

void moveSquare(AIVecPosFromPosTo newPos);

void setFraction(\_FRACTION\_INT\_ fract)

{

AiFraction = fract;

}

void addToPrevPos(vecI2d curPos)

{

previousPos.push\_back(curPos);

}

std::list<AIVecPosFromPosTo>\* getPrevPosList()

{

return &prevPosList;

}

void addToPrevPosList(AIVecPosFromPosTo curPos)

{

prevPosList.push\_back(curPos);

}

bool isPosInPrevList(AIVecPosFromPosTo curPos)

{

std::list<AIVecPosFromPosTo>::iterator list\_iter = prevPosList.begin();

while (list\_iter != prevPosList.end())

{

if ((\*list\_iter) == curPos)

{

return true;

}

}

return false;

}

void deletePosFromPrevList(AIVecPosFromPosTo curPos)

{

std::list<AIVecPosFromPosTo>::iterator iter = prevPosList.begin();

while (iter!=prevPosList.end())

{

if ((\*iter).operator==(curPos))

{

prevPosList.erase(iter);

return;

}

++iter;

}

}

std::vector<AIVecPosFromPosTo> showPlacesToCreateMyCquare(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* curSituation,

\_FRACTION\_INT\_ fract, std::vector<AIVecPosFromPosTo> myTeamPositions);

void addPosToPrevList(AIVecPosFromPosTo curPos)

{

prevPosList.push\_back(curPos);

}

void deletePosFromPreviosVec(vecI2d curPos)

{

for (int i = 0; i < previousPos.size(); i++)

{

if (curPos == previousPos[i])

{

previousPos.erase(previousPos.begin() + i);

}

}

}

int createMyTeamSquareValue(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* currSiation, \_FRACTION\_INT\_ myFraction);

bool isEnemySurroundedByMe(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* currSituation, \_FRACTION\_INT\_ myFract, int x, int y);

int killEnemySquareValue(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* currSiation, \_FRACTION\_INT\_ myFraction);

int calcSquareVal(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* curSituation, int X, int Y, \_FRACTION\_INT\_ frac);

std::vector<AIVecPosFromPosTo> getAllPossibleToMoveMyCells(\_AI\_INTELLIGENCE\_FRACTION\_DYNAMIC\_ARR\* currSituation,

\_FRACTION\_INT\_ myTeam);

static const int \_AI\_MY\_TEAM\_SQUARES\_VALUE = 10;

static const int \_AI\_MY\_TEAM\_SQUARES\_COUNT\_MODIFIER = 2;

static const int \_AI\_ENEMY\_TEAM\_SQUARES\_VALUE = 10;

static const int \_AI\_ENEMY\_TEAM\_SQUARES\_COUNT\_MODIFIER = 3;

static const int \_AI\_RECURSION\_LEAF = 4;

static const int \_AI\_RECURSION\_CAN\_CREATE\_MY\_SQUARE = 400;

static const int \_Ai\_RECURSION\_CAN\_KILL\_ENEMY = 600;

};

typedef AIRecursionData AIAnalysisData;

#endif // !\_AI\_RECURSION\_DATA\_H

**ArtificialIntelligence.cpp**

#include "ArtificialIntelligence.h"

#include "GameSimStructs.h"

#include <math.h>

#include <limits>

#define brk break

#include "Menu.h"

#include <random>

void AI::intelligenceInit()

{

int x = intelCon.getGSimDataRef()->getArrTemplInterface()->getSizeHorizonal();

int y = intelCon.getGSimDataRef()->getArrTemplInterface()->getSizeVertical();

if (x <= 0 || y <= 0)

{

throw "NO\_WALUES";

}

getCursorFrom()->setTopLeftLim(vecI2d(0, 0));

getCursorFrom()->setBotRightLim(vecI2d(x, y));

getCursorTo()->setTopLeftLim(vecI2d(0, 0));

getCursorTo()->setBotRightLim(vecI2d(x, y));

switch(getFraction())

{

case \_BFL\_TEAM\_UP\_SIDE:

{

getCursorFrom()->setCursor(0, 0);

getCursorTo()->setCursor(0,0);

brk;

}

case \_BFL\_TEAM\_DOWN\_SIDE:

{

getCursorFrom()->setCursor(x - 1, y - 1);

getCursorTo()->setCursor(x - 1, y - 1);

brk;

}

default:

break;

}

getRecurseData()->setFraction(getFraction());

getRecurseData()->curSituation.createNewArray(getArr()->getSizeHorizonal(), getArr()->getSizeVertical());

}

AIRecursionData AI::convertGSimArrToRecursionData(OEDataBase\* dbref)

{

DynamicIntArray2DTemplated<GameBaseObject>\* arrRef;

arrRef = dbref->getGameSimDataPtr()->getArrTemplInterface();

arrRef->getSizeVertical();

for (int y = 0; y < arrRef->getSizeVertical(); y++)

{

for (int x = 0; x < arrRef->getSizeHorizonal(); x++)

{

int swtch;

try

{

swtch = std::stoi(\*arrRef->getElem(x, y)->getGBOFreeDataPtr());

}

catch (std::invalid\_argument)

{

//intelCon.directLogAppend("Ai conv err");

getRecurseData()->curSituation.setElem(\_BFL\_TEAM\_NONE, x, y);

continue;

}

switch (swtch)

{

case \_BFL\_TEAM\_UP\_SIDE:

{

getRecurseData()->curSituation.setElem(\_BFL\_TEAM\_UP\_SIDE, x, y);

break;

}

case \_BFL\_TEAM\_DOWN\_SIDE:

{

getRecurseData()->curSituation.setElem(\_BFL\_TEAM\_DOWN\_SIDE, x, y);

break;

}

case \_BFL\_TEAM\_NONE:

{

getRecurseData()->curSituation.setElem(\_BFL\_TEAM\_NONE, x, y);

break;

}

default:

break;

}

}

}

return \*getRecurseData();

}

class TempMinMaxValues

{

public:

int value;

AIVecPosFromPosTo pos;

};

class ValuesCount

{

public:

ValuesCount(int val, int valCount) : value(val), valueCount(valCount){}

int value;

int valueCount;

};

void AI::chooseAction()

{

convertGSimArrToRecursionData();

std::vector<AIVecPosFromPosTo> possibleTurns = getRecurseData()->getAllPossibleTurns(playerFraction);

std::vector<vecI2d> enemiesPoses;

int index = 0;

std::vector<int> maxValuesIndexes;

int max = std::numeric\_limits<int>::min();

std::vector<AIVecPosFromPosTo>::iterator iter = possibleTurns.begin();

while (iter != possibleTurns.end())

{

int tempX = (\*iter).posTo.getX(), tempY = (\*iter).posTo.getY();

AIRecursionData innerRec;

innerRec = \*getRecurseData();

AIVecPosFromPosTo it = (\*iter);

int iterVal;

if (it.action == \_BFL\_INTELLIGENCE\_CREATE\_CQUARE)

{

iterVal = innerRec.createMyTeamSquareValue(&(innerRec.curSituation), playerFraction);

}

else

{

innerRec.moveSquare(\*iter);

iterVal = getRecurseData()->calcSquareVal(&(innerRec.curSituation), 0, 0, playerFraction);

}

if (max < iterVal)

{

if (!maxValuesIndexes.empty())

{

maxValuesIndexes.clear();

}

max = iterVal;

maxValuesIndexes.push\_back(index);

}

else if (max == iterVal)

{

maxValuesIndexes.push\_back(index);

}

index++;

++iter;

}

AIVecPosFromPosTo retVal;

std::random\_device randomiser;

std::mt19937 gener(randomiser());

if (maxValuesIndexes.size() - 1 < 0)

{

return;

}

std::uniform\_int\_distribution<> dist(0, maxValuesIndexes.size()-1);

int randIndex = maxValuesIndexes.at(dist(gener));

AIVecPosFromPosTo randomVec = possibleTurns.at(randIndex);

intelCon.playerCursorFrom->setCursor(randomVec.posFrom);

intelCon.playerCursorTo->setCursor(randomVec.posTo);

if (randomVec.action == \_BFL\_INTELLIGENCE\_CREATE\_CQUARE)

{

intelCon.playerCursorFrom->setCursor(randomVec.posTo);

intelCon.playerCursorTo->setCursor(randomVec.posTo);

setAction(\_BFL\_INTELLIGENCE\_CREATE\_CQUARE);

return;

}

setAction(\_BFL\_INTELLIGENCE\_MOVE\_CELL\_CUSTOM);

}

#undef brk

**ArtificialIntelligence.h**

#ifndef \_ARTIFICIAL\_INTELLIGENCE\_H

#define \_ARTIFICIAL\_INTELLIGENCE\_H

#include "Vector2DTemplate.h"

#include "AIRecursionData.h"

#include "Intelligence.h"

class ArtificialIntelligence : public Intelligence

{

private:

int prevListSize = 0;

AIRecursionData recurseData;

public:

ArtificialIntelligence(OEDataBase\* p\_OEDataBase,

DBIntelligenceData\* dataRef, \_FRACTION\_INT\_ fraction, BattleForLifeRules\* rules) :

Intelligence(p\_OEDataBase, dataRef, fraction, rules){}

AIRecursionData\* getRecurseData()

{

return &recurseData;

}

int MinMax(AIRecursionData recursionPos, AIVecPosFromPosTo turnPos, int level, bool isMax, \_FRACTION\_INT\_ myTeam);

void intelligenceInit();

void chooseAction();

AIRecursionData convertGSimArrToRecursionData(OEDataBase\* dbref);

AIRecursionData convertGSimArrToRecursionData()

{

return convertGSimArrToRecursionData(intelCon.getDBReference());

}

~ArtificialIntelligence(){}

bool isSquareInMyTeam(int, int) { return false; }

};

typedef ArtificialIntelligence AI;

#endif // !\_ARTIFICIAL\_INTELLIGENCE

BattleForLifeGameComponents.cpp

#include "BattleForLifeGameComponents.h"

#include <string>

#include "ArtificialIntelligence.h"

bool BattleForLifeRules::isPlaceAcceptableToMove(int x, int y)

{

if (x >= gameRuleCon.getArr()->getSizeHorizonal() || x < 0)

{

return false;

}

if (y >= gameRuleCon.getArr()->getSizeVertical() || y < 0)

{

return false;

}

if (!gameRuleCon.getArr()->getElem(x, y)->getGBOTypePtr()->\_Equal(\_GBC\_OBJECT\_TYPE))

{

return false;

}

return true;

}

#define NOT !

int BattleForLifeRules::countSquaresOfOneFraction(\_FRACTION\_INT\_ curFraction)

{

int counter = 0;

for (int y = 0; y < getBFTRulesCon()->getArr()->getSizeVertical(); y++)

{

for (int x = 0; x < getBFTRulesCon()->getArr()->getSizeHorizonal(); x++)

{

GameBaseObject tempObj;

tempObj = \*getBFTRulesCon()->getArr()->getElem(x, y);

if ((NOT tempObj.getGBOTypePtr()->\_Equal(\_GBC\_OBJECT\_TYPE))

&& tempObj.getGBOFreeDataPtr()->\_Equal(std::to\_string(curFraction)))

{

counter++;

}

}

}

return counter;

}

bool BattleForLifeRules::moveCell(\_FRACTION\_INT\_ curFraction, int curX, int curY, int newX, int newY)

{

if (NOT BattleForLifeRules::isPlaceAcceptableToMove(newX, newY))

{

return false;

}

GameBaseObject curObject;

curObject = \*gameRuleCon.getArr()->getElem(curX, curY);

if (NOT curObject.getGBOFreeDataPtr()->\_Equal(std::to\_string(curFraction)))

{

return false;

}

gameRuleCon.getArr()->setElem(curObject, newX, newY);

gameRuleCon.getArr()->setElem(GameBaseObject(), curX, curY);

return true;

}

/\*

функция показывает все доступные точки для перемещения.

Если рядом стоит клетка, то проверяется можно ли через неё перешагнуть

\*/

std::vector<vecI2d> BattleForLifeRules::showAllPosiblePositions(int x, int y)

{

std::vector<vecI2d> retVal;

if (x >= gameRuleCon.getArr()->getSizeHorizonal() || x < 0)

{

retVal.push\_back(\_BFL\_RULES\_POSSIBLE\_POS\_NOT\_FOUND\_VEC);

return retVal;

}

if (y >= gameRuleCon.getArr()->getSizeVertical() || y < 0)

{

retVal.push\_back(\_BFL\_RULES\_POSSIBLE\_POS\_NOT\_FOUND\_VEC);

return retVal;

}

#define UP\_FROM\_CURR y - \_NEIGHBOUR\_NEAR\_POSITION

//UP FROM CURR POSITION

if (isPlaceAcceptableToMove(x, UP\_FROM\_CURR))

{

retVal.push\_back(vecI2d(x, UP\_FROM\_CURR));

}

else

{

if (isPlaceAcceptableToMove(x, UP\_FROM\_CURR - \_NEIGHBOUR\_NEAR\_POSITION))

{

retVal.push\_back(vecI2d(x, UP\_FROM\_CURR - \_NEIGHBOUR\_NEAR\_POSITION));

}

}

#undef UP\_FROM\_CURR

#define DOWN\_FROM\_CURR y + \_NEIGHBOUR\_NEAR\_POSITION

//DOWN FROM CURR POSITION

if (isPlaceAcceptableToMove(x, DOWN\_FROM\_CURR))

{

retVal.push\_back(vecI2d(x, DOWN\_FROM\_CURR));

}

else

{

if (isPlaceAcceptableToMove(x, DOWN\_FROM\_CURR + \_NEIGHBOUR\_NEAR\_POSITION))

{

retVal.push\_back(vecI2d(x, DOWN\_FROM\_CURR + \_NEIGHBOUR\_NEAR\_POSITION));

}

}

#undef DOWN\_FROM\_CURR

#define LEFT\_FROM\_CURR x - \_NEIGHBOUR\_NEAR\_POSITION

//LEFT FROM CURR POSITION

if (isPlaceAcceptableToMove(LEFT\_FROM\_CURR, y ))

{

retVal.push\_back(vecI2d(LEFT\_FROM\_CURR, y));

}

else

{

if (isPlaceAcceptableToMove(LEFT\_FROM\_CURR - \_NEIGHBOUR\_NEAR\_POSITION, y))

{

retVal.push\_back(vecI2d(LEFT\_FROM\_CURR - \_NEIGHBOUR\_NEAR\_POSITION, y));

}

}

#undef LEFT\_FROM\_CURR

#define RIGHT\_FROM\_CURR x + \_NEIGHBOUR\_NEAR\_POSITION

//RIGHT FROM CURR POSITION

if (isPlaceAcceptableToMove(RIGHT\_FROM\_CURR, y))

{

retVal.push\_back(vecI2d(RIGHT\_FROM\_CURR, y));

}

else

{

if (isPlaceAcceptableToMove(RIGHT\_FROM\_CURR + \_NEIGHBOUR\_NEAR\_POSITION, y))

{

retVal.push\_back(vecI2d(RIGHT\_FROM\_CURR + \_NEIGHBOUR\_NEAR\_POSITION, y));

}

}

#undef RIGHT\_FROM\_CURR

retVal.push\_back(\_BFL\_RULES\_POSSIBLE\_POS\_NOT\_FOUND\_VEC);

if (retVal[0] == \_BFL\_RULES\_POSSIBLE\_POS\_NOT\_FOUND\_VEC)

{

return retVal;

}

retVal.pop\_back();

return retVal;

}

bool BattleForLifeRules::isGameOver()

{

AIAnalysisData helpData;

int freeSquares = 0;

const int \_MINIMUM\_EMPTY\_SQUARES = 5;

helpData.curSituation.createNewArray(getBFTRulesCon()->getArr()->getSizeHorizonal(),

getBFTRulesCon()->getArr()->getSizeVertical());

int upTeam = 0, downTeam = 0;

for (int y = 0; y < getBFTRulesCon()->getArr()->getSizeVertical(); y++)

{

for (int x = 0; x < getBFTRulesCon()->getArr()->getSizeHorizonal(); x++)

{

GameBaseObject elem = \*getBFTRulesCon()->getArr()->getElem(x, y);

if (elem.getGBOFreeDataPtr()->\_Equal(std::to\_string(\_BFL\_TEAM\_NONE)))

{

freeSquares++;

}

int swtch;

try

{

swtch = std::stoi(\*getBFTRulesCon()->getArr()->getElem(x, y)->getGBOFreeDataPtr());

}

catch (std::invalid\_argument)

{

//intelCon.directLogAppend("Ai conv err");

helpData.curSituation.setElem(\_BFL\_TEAM\_NONE, x, y);

continue;

}

switch (swtch)

{

case \_BFL\_TEAM\_UP\_SIDE:

{

helpData.curSituation.setElem(\_BFL\_TEAM\_UP\_SIDE, x, y);

upTeam++;

break;

}

case \_BFL\_TEAM\_DOWN\_SIDE:

{

helpData.curSituation.setElem(\_BFL\_TEAM\_DOWN\_SIDE, x, y);

downTeam++;

break;

}

case \_BFL\_TEAM\_NONE:

{

helpData.curSituation.setElem(\_BFL\_TEAM\_NONE, x, y);

break;

}

default:

break;

}

}

}

if (freeSquares > \_MINIMUM\_EMPTY\_SQUARES)

{

return false;

}

//helpData

helpData.AiFraction = \_BFL\_TEAM\_UP\_SIDE;

std::vector<AIVecPosFromPosTo> allTurnsUp;

allTurnsUp = helpData.getAllPossibleTurns(helpData.AiFraction);

std::vector<AIVecPosFromPosTo> allTurnsDown;

helpData.AiFraction = \_BFL\_TEAM\_DOWN\_SIDE;

allTurnsUp = helpData.getAllPossibleTurns(helpData.AiFraction);

int turnsCount = + allTurnsUp.size();

if (0 == turnsCount || downTeam == 0 || upTeam == 0)

{

if (upTeam > downTeam)

{

gameRuleCon.getDBReference()->getMenuData()->setLosedFraction(\_BFL\_TEAM\_DOWN\_SIDE);

}

else if (upTeam < downTeam)

{

gameRuleCon.getDBReference()->getMenuData()->setLosedFraction(\_BFL\_TEAM\_UP\_SIDE);

}

else

{

gameRuleCon.getDBReference()->getMenuData()->setLosedFraction(\_BFL\_TEAM\_NONE);

}

return true;

}

return false;

}

//проверяет команду и записывает в счетчик

void fractionCheckSTR(std::string curFractionStr, unsigned short int\* counter1, unsigned short int\* counter2)

{

\_FRACTION\_INT\_ curFrac = 10;

try

{

curFrac = std::stoi(curFractionStr);

}

catch (std::invalid\_argument& invArg)

{

return;

}

catch (std::out\_of\_range)

{

return;

}

switch (curFrac)

{

case \_BFL\_TEAM\_UP\_SIDE:

{

(\*counter1)++;

break;

}

case \_BFL\_TEAM\_DOWN\_SIDE:

{

(\*counter2)++;

break;

}

default:

break;

}

}

#define shortLimitCheck gameRuleCon.getArr()->isCoordInLimits

//Проверка какие клетки откружают текущую по координатам

\_FRACTION\_INT\_ BattleForLifeRules::whatFractionSurrounds(int x, int y)

{

//vecI2d curPos(x, y);

if (NOT shortLimitCheck(x, y))

{

return \_BFL\_TEAM\_NONE;

}

unsigned short int count = 0;

unsigned short int fractionOne = 0;

unsigned short int fractionTwo = 0;

GameBaseObject tempElem;

if (shortLimitCheck(x, y - 1)

&& gameRuleCon.getArr()->getElem(x, y - 1)->getGBOTypePtr()->\_Equal(\_BFL\_TYPE\_CREATURE\_SQUARE))

{

fractionCheckSTR(\*gameRuleCon.getArr()->getElem(x, y - 1)->getGBOFreeDataPtr(),

&fractionOne, &fractionTwo

);

}

if (shortLimitCheck(x, y + 1)

&& gameRuleCon.getArr()->getElem(x, y + 1)->getGBOTypePtr()->\_Equal(\_BFL\_TYPE\_CREATURE\_SQUARE))

{

fractionCheckSTR(\*gameRuleCon.getArr()->getElem(x, y + 1)->getGBOFreeDataPtr(),

&fractionOne, &fractionTwo

);

}

if (shortLimitCheck(x - 1, y)

&& gameRuleCon.getArr()->getElem(x - 1, y)->getGBOTypePtr()->\_Equal(\_BFL\_TYPE\_CREATURE\_SQUARE))

{

fractionCheckSTR(\*gameRuleCon.getArr()->getElem(x - 1, y)->getGBOFreeDataPtr(),

&fractionOne, &fractionTwo

);

}

if (shortLimitCheck(x + 1, y)

&& gameRuleCon.getArr()->getElem(x + 1, y)->getGBOTypePtr()->\_Equal(\_BFL\_TYPE\_CREATURE\_SQUARE))

{

fractionCheckSTR(\*gameRuleCon.getArr()->getElem(x + 1, y)->getGBOFreeDataPtr(),

&fractionOne, &fractionTwo

);

}

//Выдача каких клеток больше

if (fractionOne >= \_BFL\_RULES\_NECESSARY\_SQUARES)

{

return \_BFL\_TEAM\_UP\_SIDE;

}

if (fractionTwo >= \_BFL\_RULES\_NECESSARY\_SQUARES)

{

return \_BFL\_TEAM\_DOWN\_SIDE;

}

return \_BFL\_TEAM\_NONE;

}

//функция уничтожение клетки, окруженной врагами

void BattleForLifeRules::destroySquareInSurroundings()

{

int maxX = gameRuleCon.getArr()->getSizeHorizonal();

int maxY = gameRuleCon.getArr()->getSizeVertical();

for (int y = 0; y < maxY; y++)

{

for (int x = 0; x < maxX; x++)

{

switch (whatFractionSurrounds(x, y))

{

case \_BFL\_TEAM\_NONE:

{

break;

}

case \_BFL\_TEAM\_UP\_SIDE:

{

if (gameRuleCon.getArr()->getElem(x, y)->getGBOFreeDataPtr()->\_Equal(std::to\_string(\_BFL\_TEAM\_DOWN\_SIDE)))

{

gameRuleCon.getArr()->setElem(GameBaseObject(), x, y);

}

break;

}

case \_BFL\_TEAM\_DOWN\_SIDE:

{

if (gameRuleCon.getArr()->getElem(x, y)->getGBOFreeDataPtr()->\_Equal(std::to\_string(\_BFL\_TEAM\_UP\_SIDE)))

{

gameRuleCon.getArr()->setElem(GameBaseObject(), x, y);

}

break;

}

default:

break;

}

}

}

}

//Создание клетки в текущей позии

bool BattleForLifeRules::createSquareByPressintEnter()

{

\_FRACTION\_INT\_ fractionTurn = gameRuleCon.getGSimCon()->getGameSimDataRef()->getCurrentFractionTurn();

vecI2d curFractionSquarePos;

switch (fractionTurn)

{

case \_BFL\_TEAM\_UP\_SIDE:

{

curFractionSquarePos = gameRuleCon.getDBReference()->getPlayerTwoData()->getCursorTo()->getCursorPos();

break;

}

case \_BFL\_TEAM\_DOWN\_SIDE:

{

curFractionSquarePos = gameRuleCon.getDBReference()->getPlayerOneData()->getCursorTo()->getCursorPos();

break;

}

default:

break;

}

\_FRACTION\_INT\_ fraction = whatFractionSurrounds(

//GSimCon->getDBReference()->getIntelligenceData()->getCursorTo()->getCursorPos()

curFractionSquarePos

);

switch (fraction)

{

case \_BFL\_TEAM\_UP\_SIDE:

{

gameRuleCon.getArr()->setElem(createSquareWithTeam(\_BFL\_TEAM\_UP\_SIDE),

gameRuleCon.getDBReference()->getPlayerTwoData()->getCursorTo()->getCursorPos().getX(),

gameRuleCon.getDBReference()->getPlayerTwoData()->getCursorTo()->getCursorPos().getY()

);

return true;

break;

}

case \_BFL\_TEAM\_DOWN\_SIDE:

{

gameRuleCon.getArr()->setElem(createSquareWithTeam(\_BFL\_TEAM\_DOWN\_SIDE),

gameRuleCon.getDBReference()->getPlayerOneData()->getCursorTo()->getCursorPos().getX(),

gameRuleCon.getDBReference()->getPlayerOneData()->getCursorTo()->getCursorPos().getY()

);

return true;

break;

}

{

default:

break;

}

}

return false;

}

GameBaseObject BattleForLifeRules::createSquareWithTeam(\_FRACTION\_INT\_ fraction)

{

GameBaseObject retObj;

retObj.setGBOName(\_BFL\_STANDART\_NAME);

retObj.setGBOType(\_BFL\_TYPE\_CREATURE\_SQUARE);

retObj.setGBOFreeData(std::to\_string(fraction));

return retObj;

}

#undef shortLimitCheck

#undef NOT

**BattleForLifeGameComponents.h**

#ifndef BATTLEFORLIFECOMPONENTS\_H\_

#define BATTLEFORLIFECOMPONENTS\_H\_

#include "GameBaseComponents.h"

#include "GameSimConnector.h"

#include <vector>

#define \_BFL\_TYPE\_CREATURE\_SQUARE "creatureSquare"

#define \_BFL\_STANDART\_NAME "creatureSquareName"

#define \_BFL\_TEAM\_ERR\_NO\_FRACTION\_CHOSEN "BFL\_TEAM\_ERR\_NO\_FRACTION\_CHOSEN"

#define \_BFL\_TEAM\_NONE 0

#define \_BFL\_TEAM\_UP\_SIDE 2

#define \_BFL\_TEAM\_DOWN\_SIDE 1

#define \_BFL\_RULES\_NECESSARY\_SQUARES 3

#define \_FRACTION\_INT\_ short unsigned int

/\*

Коннектор игровых правил с БД

\*/

class BattleForLifeConnector : public OEDBConnector

{

private:

GameSimConnector\* GSimCon;

DynamicIntArray2DTemplated<GameBaseObject>\* gObjArr;

public:

BattleForLifeConnector(GameSimConnector\* GSimConRef) : OEDBConnector(GSimConRef->getDBReference()), GSimCon(GSimConRef)

{}

DynamicIntArray2DTemplated<GameBaseObject>\* getArr()

{

return getDBReference()->getGameSimDataPtr()->getArrTemplInterface();

}

GameSimConnector\* getGSimCon()

{

return GSimCon;

}

};

#define \_BFL\_RULES\_POSSIBLE\_POS\_NOT\_FOUND -2

/\* || vecI2d(\_BFL\_RULES\_POSSIBLE\_POS\_NOT\_FOUND, \_BFL\_RULES\_POSSIBLE\_POS\_NOT\_FOUND) || \*/

#define \_BFL\_RULES\_POSSIBLE\_POS\_NOT\_FOUND\_VEC vecI2d(\_BFL\_RULES\_POSSIBLE\_POS\_NOT\_FOUND, \_BFL\_RULES\_POSSIBLE\_POS\_NOT\_FOUND)

/\*

Класс правил игры "Борьба за жизнь"

Содержатся игровые действия и события

Класс выполняет действия над реальными объектами по условиям правил игры

\*/

class BattleForLifeRules

{

private:

BattleForLifeConnector gameRuleCon; //коннектор

\_FRACTION\_INT\_ fractionTurn = \_BFL\_TEAM\_NONE; //текущая команда на данном ходу

public:

BattleForLifeRules(GameSimConnector\* GameSimCon) : gameRuleCon(GameSimCon)

{

std::cout << "Rules Constr\n";

}

BattleForLifeConnector\* getBFTRulesCon()

{

return &gameRuleCon;

}

GameSimConnector\* getGSimCon()

{

return gameRuleCon.getGSimCon();

}

bool isPlaceAcceptableToMove(int X, int Y); //доступно ли место для перемещения

void setCurrPlayingFraction(\_FRACTION\_INT\_ currentFraction)

{

fractionTurn = currentFraction;

}

\_FRACTION\_INT\_ getCurrPlayingFraction()

{

return fractionTurn;

}

void swapFraction() //меняем текущую команду

{

switch (getCurrPlayingFraction())

{

case \_BFL\_TEAM\_DOWN\_SIDE:

{

fractionTurn = \_BFL\_TEAM\_UP\_SIDE;

break;

}

case \_BFL\_TEAM\_UP\_SIDE:

{

fractionTurn = \_BFL\_TEAM\_DOWN\_SIDE;

break;

}

case \_BFL\_TEAM\_NONE:

{

throw \_BFL\_TEAM\_ERR\_NO\_FRACTION\_CHOSEN;

}

default:

break;

}

}

bool moveCell(\_FRACTION\_INT\_ curFraction, int curX, int curY, int newX, int newY); //двигаем клетку по координатам

bool moveCell(\_FRACTION\_INT\_ curFraction, vecI2d vecFrom, vecI2d vecTo)//двигаем клетку по координатам векторов

{

return moveCell(

//

curFraction, vecFrom.getX(), vecFrom.getY(), vecTo.getX(), vecTo.getY()

//

);

}

//0 - //Nothing surrounds// || 1 - \_BFL\_TEAM\_DOWN\_SIDE || 2 - \_BFL\_TEAM\_UP\_SIDE

\_FRACTION\_INT\_ whatFractionSurrounds(int X, int Y); //проверяем какие клетки окружают текущую

//0 - //Nothing surrounds// || 1 - \_BFL\_TEAM\_DOWN\_SIDE || 2 - \_BFL\_TEAM\_UP\_SIDE

\_FRACTION\_INT\_ whatFractionSurrounds(VectorInt2D currentPosition)

{

return whatFractionSurrounds(

currentPosition.getX()

,

currentPosition.getY()

);

}

bool createSquareByPressintEnter(); //Создание новой клетки, при соблюдении условий

GameBaseObject createSquareWithTeam(\_FRACTION\_INT\_ fraction); //Созданием клетки с указанием команды

void destroySquareInSurroundings(); //уничтожение клетки в окружении врагов

int countSquaresOfOneFraction(\_FRACTION\_INT\_ curFraction); //посчитать все клетки из одной команды

bool isGameOver();

std::vector<vecI2d> showAllPosiblePositions(int X, int Y); //выдать все доступные точки для перемещения

std::vector<vecI2d> showAllPosiblePositions(VectorInt2D currentPosition)

{

return showAllPosiblePositions(

currentPosition.getX()

,

currentPosition.getY()

);

}

const int \_NEIGHBOUR\_NEAR\_POSITION = 1;

};

#endif // !BATTLEFORLIFECOMPONENTS\_H\_

**GameBaseComponents.cpp**

#include "GameBaseComponents.h"

GameBaseObject::GameBaseObject()

{

setGBOName(\_GBC\_OBJECT\_STANDART\_NAME);

setGBOType(\_GBC\_OBJECT\_TYPE);

setGBOFreeData(\_GBC\_OBJECT\_STANDART\_FREE\_DATA);

}

GameBaseObject::~GameBaseObject()

{

getGBONamePtr()->clear();

getGBOTypePtr()->clear();

getGBOFreeDataPtr()->clear();

}

VectorInt2D::VectorInt2D()

{

setX(\_GBO\_VECTOR\_2D\_NOT\_SET);

setY(\_GBO\_VECTOR\_2D\_NOT\_SET);

}

VectorInt2D::VectorInt2D(int X, int Y)

{

setX(X);

setY(Y);

}

VectorInt2D::~VectorInt2D()

{

}

Cursor::Cursor(VectorInt2D startPos)

{

curPos = vecI2d(startPos);

topAndLeftLimiter.setVectorI2D(\_\_GBO\_CURSOR\_LIMIT\_NOT\_SET, \_\_GBO\_CURSOR\_LIMIT\_NOT\_SET);

BottomAndRightLimiter.setVectorI2D(\_\_GBO\_CURSOR\_LIMIT\_NOT\_SET, \_\_GBO\_CURSOR\_LIMIT\_NOT\_SET);

}

Cursor::Cursor()

{

curPos = vecI2d(\_\_GBO\_CURSOR\_START\_POS\_NOT\_SET, \_\_GBO\_CURSOR\_START\_POS\_NOT\_SET);

topAndLeftLimiter.setVectorI2D(\_\_GBO\_CURSOR\_LIMIT\_NOT\_SET, \_\_GBO\_CURSOR\_LIMIT\_NOT\_SET);

BottomAndRightLimiter.setVectorI2D(\_\_GBO\_CURSOR\_LIMIT\_NOT\_SET, \_\_GBO\_CURSOR\_LIMIT\_NOT\_SET);

}

void Cursor::setCursor(int X, int Y)

{

if (isAcceptableToMoveCursor(X, Y))

{

curPos.setVectorI2D(X, Y);

}

}

void Cursor::setCursor(vecI2d newPos)

{

setCursor(

newPos.getX()

,

newPos.getY()

);

}

bool Cursor::isLimitsSet()

{

if (\_\_GBO\_CURSOR\_LIMIT\_NOT\_SET == topAndLeftLimiter.getX() || \_\_GBO\_CURSOR\_LIMIT\_NOT\_SET == BottomAndRightLimiter.getX())

{

return false;

}

if (\_\_GBO\_CURSOR\_LIMIT\_NOT\_SET == topAndLeftLimiter.getY() || \_\_GBO\_CURSOR\_LIMIT\_NOT\_SET == BottomAndRightLimiter.getY())

{

return false;

}

return true;

}

bool Cursor::isAcceptableToMoveCursor(int x, int y)

{

if (!isLimitsSet())

{

throw \_\_GBO\_CURSOR\_ERR\_LIMIT\_NOT\_SET;

}

if (x >= BottomAndRightLimiter.getX() || x < topAndLeftLimiter.getX())

{

return false;

}

if (y >= BottomAndRightLimiter.getY() || y < topAndLeftLimiter.getY())

{

return false;

}

return true;

}

void Cursor::crsUP()

{

if (curPos == vecI2d(\_\_GBO\_CURSOR\_START\_POS\_NOT\_SET, \_\_GBO\_CURSOR\_START\_POS\_NOT\_SET))

{

throw \_\_GBO\_CURSOR\_ERR\_NO\_POSITION\_SET;

}

int curX = curPos.getX();

int curY = curPos.getY();

if (isAcceptableToMoveCursor(curX, curY - 1))

{

curPos.setY(curY - 1);

}

}

void Cursor::crsDOWN()

{

int curX = curPos.getX();

int curY = curPos.getY();

if (isAcceptableToMoveCursor(curX, curY + 1))

{

curPos.setY(curY + 1);

}

}

void Cursor::crsLEFT()

{

int curX = curPos.getX();

int curY = curPos.getY();

if (isAcceptableToMoveCursor(curX - 1, curY))

{

curPos.setX(curX - 1);

}

}

void Cursor::crsRIGHT()

{

int curX = curPos.getX();

int curY = curPos.getY();

if (isAcceptableToMoveCursor(curX + 1, curY))

{

curPos.setX(curX + 1);

}

}

**GameBaseComponents.h**

#ifndef GAMEBASECOMPONENTS\_H\_

#define GAMEBASECOMPONENTS\_H\_

#include <string>

#include <iostream>

#include <vector>

#define \_GBC\_OBJECT\_TYPE "GameBaseObject"

#define \_GBC\_OBJECT\_STANDART\_NAME "NoGameObjectName"

#define \_GBC\_OBJECT\_STANDART\_FREE\_DATA "NoFreeData"

class GameBaseObject;

typedef GameBaseObject GBO;

class VectorInt2D;

typedef VectorInt2D vecI2d;

class DynamicIntArray2D;

typedef DynamicIntArray2D DIA2D;

#define GBC\_TYPE\_VECINT2D "VectorInt2D"

/\*

класс игровой клетки

\*/

class GameBaseObject

{

public:

GameBaseObject();

~GameBaseObject();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void setGBOName(std::string NewName) { GBOName = NewName; }

std::string\* getGBONamePtr() { return &GBOName; }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void setGBOType(std::string NewType) { GBOType = NewType; }

std::string\* getGBOTypePtr() { return &GBOType; }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void setGBOFreeData(std::string NewFreeData) { GBOFreeData = NewFreeData; }

std::string\* getGBOFreeDataPtr() { return &GBOFreeData; }

protected:

std::string GBOName;

std::string GBOType; //Строковый тип

std::string GBOFreeData; //Дополнительная строковая информация (используется для храния команды игрока)

};

#define \_GBO\_VECTOR\_2D\_NOT\_SET -2

#define \_GBO\_VECTOR\_ERR\_VEC\_NOT\_INIT "GBO\_VECTOR\_ERR\_VEC\_NOT\_INIT"

/\*

Класс математического вектора с несколькими операциями над ним

\*/

class VectorInt2D

{

public:

VectorInt2D();

VectorInt2D(int X, int Y);

bool operator ==(VectorInt2D compareVec)

{

if (this->getX() == compareVec.getX() && this->getY() == compareVec.getY())

{

return true;

}

return false;

}

bool operator !=(VectorInt2D compareVec)

{

if (this->getX() == compareVec.getX() && this->getY() == compareVec.getY())

{

return false;

}

return true;

}

std::string getStrCoords()

{

return "X: " + std::to\_string(getX()) + "\tY: " + std::to\_string(getY());

}

~VectorInt2D();

void setX(int newCoordX) { vecCoordX = newCoordX; }

int getX()

{

return vecCoordX;

}

void setY(int newCoordY) { vecCoordY = newCoordY; }

int getY()

{

return vecCoordY;

}

void setVectorI2D(int newX, int newY) { vecCoordX = newX; vecCoordY = newY; }

void setVectorI2D(vecI2d newVec) { vecCoordX = newVec.getX(); vecCoordY = newVec.getY(); }

protected:

int vecCoordX = \_GBO\_VECTOR\_2D\_NOT\_SET;

int vecCoordY = \_GBO\_VECTOR\_2D\_NOT\_SET;

};

#define \_\_GBO\_CURSOR\_LIMIT\_NOT\_SET -1

#define \_\_GBO\_CURSOR\_START\_POS\_NOT\_SET -2

#define \_\_GBO\_VECTOR\_GSIM\_OUT\_OF\_RANGE -3

#define \_\_GBO\_VECTOR\_GSIM\_OUT\_OF\_RANGE\_VEC vecI2d(\_\_GBO\_VECTOR\_GSIM\_OUT\_OF\_RANGE, \_\_GBO\_VECTOR\_GSIM\_OUT\_OF\_RANGE)

#define \_\_GBO\_CURSOR\_ERR\_LIMIT\_NOT\_SET "GBO\_CURSOR\_ERR\_LIMIT\_NOT\_SET"

#define \_\_GBO\_CURSOR\_ERR\_NO\_POSITION\_SET "GBO\_CURSOR\_ERR\_NO\_POSITION\_SET"

#define \_\_GBO\_CURSOR\_VEC\_NOT\_SET vecI2d(\_\_GBO\_CURSOR\_START\_POS\_NOT\_SET, \_\_GBO\_CURSOR\_START\_POS\_NOT\_SET)

/\*

Класс курсора

Курсор - показывает местоположение клетки и позволяет работать точечно с ней

Содержатся действия над курсором

\*/

class Cursor

{

private:

VectorInt2D curPos, //позиция курсора

topAndLeftLimiter, //ограничитель сверху и слева

BottomAndRightLimiter; //ограничитель снизу и справа

public:

Cursor(VectorInt2D startPos);

Cursor();

vecI2d getCursorPos()

{

return curPos;

}

void setCursor(int X, int Y);

void setCursor(vecI2d newPos);

vecI2d\* getCursorPtr()

{

return &curPos;

}

bool isLimitsSet();

void setTopLeftLim(vecI2d lim)

{

topAndLeftLimiter = vecI2d(lim);

}

void setBotRightLim(vecI2d lim)

{

BottomAndRightLimiter = vecI2d(lim);

}

void crsUP();

void crsDOWN();

void crsLEFT();

void crsRIGHT();

bool isAcceptableToMoveCursor(int x, int y);

};

#endif

**GameCore.cpp**

#include "GameCore.h"

#include "OeDataBase.h"

#include "GraphicCore.h"

#include <iostream>

#include <windows.h>

/\*

GameCore main function. There starts game

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Переменные объявлять в модуле базы данных

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Вызывать переменные только через геттеры

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Передавать в функции указатели

\*/

GCReturnValues GameCore::GCstartMainFunc(GCStartingArgs GCInpArgs)

{

GCInitiateModules(); //инициализирует во всем модулях начальные значения, которые добавляет логика игры

isGCRunning = true;

GCReturnValues GCret; //не используется

GrCoreInputArgs GrCarg; // аргументы вывода на экран для граф.ядра

GrCarg.flag = \_MENU\_PAGE\_GREETINGS;

getDBptr()->getMenuData()->setPage(\_MENU\_PAGE\_MAIN);

//здесь выводится первое сообщение на экран.

getGrCorePtr()->GrCoreInputHandler( //Вызываем обработчик входа графического ядра

getDBptr()->getGrCData(), //Закидываем указатель с данными для граф.ядра

GrCarg //аргумент запуска

);

Sleep(3000);

while (isGCRunning != false)

{

mainMenu.keyBoardInput();//запрашиваем меню для считывания команды с клавиатуры

GrCarg.flag = getDBptr()->getMenuData()->getPage();

gameSim\_ex.GSimInputHandler(); //запускаем ядро симуляции игры через обработчик ввода

mainMenu.solveCommand();

getGrCorePtr()->GrCoreInputHandler( //Вызываем обработчик входа графического ядра

getDBptr()->getGrCData(), //Закидываем указатель с данными для граф.ядра

GrCarg //аргумент запуска

);

GrCarg.flag = getDBptr()->getMenuData()->getPage();

//mainMenu.solveCommand();

getGrCorePtr()->GrCoreInputHandler( //Вызываем обработчик входа графического ядра

getDBptr()->getGrCData(), //Закидываем указатель с данными для граф.ядра

GrCarg //аргумент запуска

);

int tempInt = getDBptr()->getMenuData()->getPage();

if (tempInt == \_MENU\_PAGE\_EXIT)

{

isGCRunning = false;

GCret.isSuccess = true;

}

}

return GCret;

}

void GameCore::GCInitiateModules()

{

GrCore\_ex.GraphicCoreInitialization(getDBptr());

gameSim\_ex.GameSimulationInitialisation();

}

**GameCore.h**

#ifndef GAMECORE\_H\_

#define GAMECORE\_H\_

#include "OeDataBase.h"

#include "GraphicCoreStructs.h"

#include "GraphicCore.h"

#include "GameSimulation.h"

#include "Menu.h"

/\*

Core input params:

int: flag

\*/

struct GCStartingArgs

{

int testFlag;

};

/\*

Core ending values

int: Value

bool: isSuccess

\*/

struct GCReturnValues

{

int testRetVal;

bool isSuccess;

};

/\*

Здесь содержится объявление ядра всей игры.

Ядро содержит другие подядра и базу данныъ

\*/

class GameCore

{

public:

//starts main loop

GCReturnValues GCstartMainFunc(GCStartingArgs GCInpArgs);

void GCInitiateModules();

GameCore() : GrCore\_ex(getDBptr()) , gameSim\_ex(getDBptr()),

mainMenu(getDBptr()), DataBase()

{

isGCRunning = false;

}

GraphicCore getGrCore()

{

return GrCore\_ex;

}

GraphicCore\* getGrCorePtr()

{

return &GrCore\_ex;

}

OEDataBase\* getDBptr() { return &DataBase; }

private:

OEDataBase DataBase; //экземпляр модуля базы данных

GraphicCore GrCore\_ex; //экземпляр модуля граф.ядра

GameSimulation gameSim\_ex; //экземпляр модуля сим.игры

MenuInterface mainMenu; //экземпляр модуля меню

bool isGCRunning; // is game active.

};

#endif // !GAMECORE\_H\_

**GameSimConnector.h**

#ifndef \_GAME\_SIMULATION\_CONNECTOR\_H\_

#define \_GAME\_SIMULATION\_CONNECTOR\_H\_

#include "OeDataBase.h"

#define GAMESIMULATION\_CONNECTOR\_ERR\_NO\_DATABASE\_PTR "GAMESIMULATION\_CONNECTOR\_ERR\_NO\_DATABASE\_PTR"

class GameSimConnector;

class GameSimConnector : public OEDBConnector

{

public:

GameSimConnector(OEDataBase\* dataBasePtr) : OEDBConnector(dataBasePtr),

GameSimDataReference(getDBReference()->getGameSimDataPtr()),

menuData(getDBReference()->getMenuData())

{

GBOArrRef = getGameSimDataRef()->getArrTemplInterface();

std::cout << "game sim connector constructor\n";

}

~GameSimConnector()

{

OEDBConnector::~OEDBConnector();

}

DBGameSimData\* getGameSimDataRef()

{

return GameSimDataReference;

}

DBMenuData\* getMenuDataRef()

{

return menuData;

}

DynamicIntArray2DTemplated<GameBaseObject>\* getGBOArr()

{

return GBOArrRef;

}

int getGBOArrMaxX()

{

return getGameSimDataRef()->getArrTemplInterface()->getSizeHorizonal();

}

int getGBOArrMaxY()

{

return getGameSimDataRef()->getArrTemplInterface()->getSizeVertical();

}

protected:

DBMenuData\* menuData;

DynamicIntArray2DTemplated<GameBaseObject>\* GBOArrRef;

DBGameSimData\* GameSimDataReference;

};

#endif // !\_GAME\_SIMULATION\_CONNECTOR\_H\_

**GameSimStructs.h**

#ifndef \_GAME\_SIMULATION\_STRUCTS\_H\_

#define \_GAME\_SIMULATION\_STRUCTS\_H\_

#define \_GAMESIM\_MODE\_NO\_COMMAND 0

#define \_GAMESIM\_MODE\_MOVE\_CELL\_UP 1

#define \_GAMESIM\_MODE\_MOVE\_CELL\_DOWN 2

#define \_GAMESIM\_MODE\_MOVE\_CELL\_LEFT 3

#define \_GAMESIM\_MODE\_MOVE\_CELL\_RIGHT 4

#define \_GAMESIM\_MODE\_CELL\_ENTER 5

#define \_GAMESIM\_MODE\_CELL\_RESET 6

#define \_GSIM\_ARR\_SIZE\_X 5

#define \_GSIM\_ARR\_SIZE\_Y 5

#define \_GSIM\_ARR\_SIZE\_NOT\_SET -1

#endif // !\_GAME\_SIMULATION\_STRUCTS\_H\_

**GameSimulation.cpp**

#include "GameSimulation.h"

#include "GameBaseComponents.h"

#define brk break

#define NOT !

void GameSimulation::GameSimulationInitialisation()

{

if (NULL == GSimCon.getDBReference())

{

throw GAMESIMULATION\_CONNECTOR\_ERR\_NO\_DATABASE\_PTR;

}

int GSimSizeX, GSimSizeY;

GSimSizeX = GSimCon.getGameSimDataRef()->getGSimArrSizeHorizontal();

GSimSizeY = GSimCon.getGameSimDataRef()->getGSimArrSizeVertical();

if (GSimSizeX == \_GSIM\_ARR\_SIZE\_NOT\_SET || GSimSizeY == \_GSIM\_ARR\_SIZE\_NOT\_SET)

{

//стандартные размеры массива

GSimCreateStartBaseGameElements(\_GSIM\_ARR\_SIZE\_X, \_GSIM\_ARR\_SIZE\_Y);

}

else

{

GSimCreateStartBaseGameElements(GSimSizeX, GSimSizeY);

}

//Задает стартовые значения игры

//Инициализируем значения игрока и выьирам тим интеллекта

//humanPlaterOne.playerInit(\_PLAYER\_INTELLIGENCE\_MODE\_PERSON\_MIND); //Обычный игрок

humanPlaterOne.playerInit(\_PLAYER\_INTELLIGENCE\_MODE\_COMPUTER\_MIND); //Компьютер

//Для того, чтобы показать работоспособность основных элементов, выключаю компьютерного игрока

humanPlaterTwo.playerInit(\_PLAYER\_INTELLIGENCE\_MODE\_COMPUTER\_MIND); //и ставлю обычного

}

//Функция проверяет в зависимости от данных с меню, какие операции стоит совершить

void GameSimulation::GSimInputHandler()

{

if (isGameSim\_NOT\_FullInit())

{

return;

}

switch (GSimCon.getMenuDataRef()->getPage())

{

case \_MENU\_PAGE\_RESTART\_GAME:

{

GSimCon.getMenuDataRef()->setPage(\_MENU\_PAGE\_GAME\_SIM);

int GSimSizeX, GSimSizeY;

GSimSizeX = GSimCon.getGameSimDataRef()->getGSimArrSizeHorizontal();

GSimSizeY = GSimCon.getGameSimDataRef()->getGSimArrSizeVertical();

if (GSimSizeX == \_GSIM\_ARR\_SIZE\_NOT\_SET || GSimSizeY == \_GSIM\_ARR\_SIZE\_NOT\_SET)

{

//стандартные размеры массива

GSimCreateStartBaseGameElements(\_GSIM\_ARR\_SIZE\_X, \_GSIM\_ARR\_SIZE\_Y);

return;

}

GSimCreateStartBaseGameElements(GSimSizeX, GSimSizeY);

return;

break;

}

case \_MENU\_PAGE\_GAME\_SIM:

{

break;

}

case \_MENU\_PAGE\_PLAY\_WITH\_BOT:

{

GSimCon.getMenuDataRef()->setPage(\_MENU\_PAGE\_RESTART\_GAME);

humanPlaterOne.playerInit(\_PLAYER\_INTELLIGENCE\_MODE\_PERSON\_MIND); //Обычный игрок

humanPlaterTwo.playerInit(\_PLAYER\_INTELLIGENCE\_MODE\_COMPUTER\_MIND);

return;

}

case \_MENU\_PAGE\_PLAY\_BY\_BOTS:

{

GSimCon.getMenuDataRef()->setPage(\_MENU\_PAGE\_RESTART\_GAME);

humanPlaterOne.playerInit(\_PLAYER\_INTELLIGENCE\_MODE\_COMPUTER\_MIND);

humanPlaterTwo.playerInit(\_PLAYER\_INTELLIGENCE\_MODE\_COMPUTER\_MIND);

return;

}

case \_MENU\_PAGE\_PLAY\_PERSON\_WITH\_PERSON:

{

GSimCon.getMenuDataRef()->setPage(\_MENU\_PAGE\_RESTART\_GAME);

humanPlaterOne.playerInit(\_PLAYER\_INTELLIGENCE\_MODE\_PERSON\_MIND);

humanPlaterTwo.playerInit(\_PLAYER\_INTELLIGENCE\_MODE\_PERSON\_MIND);

return;

}

case \_MENU\_PAGE\_CHANGE\_ARR\_SIZE:

{

GSimCon.getMenuDataRef()->setPage(\_MENU\_PAGE\_MAIN);

int GSimSizeX, GSimSizeY;

GSimSizeX = GSimCon.getGameSimDataRef()->getGSimArrSizeHorizontal();

GSimSizeY = GSimCon.getGameSimDataRef()->getGSimArrSizeVertical();

if (GSimSizeX == \_GSIM\_ARR\_SIZE\_NOT\_SET || GSimSizeY == \_GSIM\_ARR\_SIZE\_NOT\_SET)

{

//стандартные размеры массива

GSimCreateStartBaseGameElements(\_GSIM\_ARR\_SIZE\_X, \_GSIM\_ARR\_SIZE\_Y);

return;

}

GSimCreateStartBaseGameElements(GSimSizeX, GSimSizeY);

return;

}

default:

return;

break;

}

int menuCase = GSimCon.getMenuDataRef()->getKeyCase();

switch (menuCase)

{

case \_MENU\_CASE\_SET\_UP:

{

GSimCon.getGameSimDataRef()->setGameSimMenuMode(\_GAMESIM\_MODE\_MOVE\_CELL\_UP);

break;

}

case \_MENU\_CASE\_SET\_DOWN:

{

GSimCon.getGameSimDataRef()->setGameSimMenuMode(\_GAMESIM\_MODE\_MOVE\_CELL\_DOWN);

break;

}

case \_MENU\_CASE\_SET\_LEFT:

{

GSimCon.getGameSimDataRef()->setGameSimMenuMode(\_GAMESIM\_MODE\_MOVE\_CELL\_LEFT);

break;

}

case \_MENU\_CASE\_SET\_RIGHT:

{

GSimCon.getGameSimDataRef()->setGameSimMenuMode(\_GAMESIM\_MODE\_MOVE\_CELL\_RIGHT);

brk;

}

case \_MENU\_CASE\_SET\_ENTER:

{

GSimCon.getGameSimDataRef()->setGameSimMenuMode(\_GAMESIM\_MODE\_CELL\_ENTER);

brk;

}

case \_MENU\_CASE\_SET\_RESET:

{

GSimCon.getGameSimDataRef()->setGameSimMenuMode(\_GAMESIM\_MODE\_CELL\_RESET);

brk;

}

case \_MENU\_CASE\_EXIT:

{

GSimCon.getMenuDataRef()->setPage(\_MENU\_PAGE\_PAUSE);

return;

}

default:

GSimCon.getGameSimDataRef()->setGameSimMenuMode(\_GAMESIM\_MODE\_NO\_COMMAND);

break;

}

GameSim::GSimStartMainFunction();

}

void GameSimulation::GSimStartMainFunction()

{

gameRule.destroySquareInSurroundings();

if (gameRule.isGameOver() == true)

{

GSimCon.getMenuDataRef()->setPage(\_MENU\_PAGE\_GAME\_OVER);

return;

}

//Каждый такт цикла проверяется какой игрок

switch (gameRule.getCurrPlayingFraction())

{

case \_BFL\_TEAM\_DOWN\_SIDE:

{

GSimCon.getGameSimDataRef()->setCurrFractionTurn(gameRule.getCurrPlayingFraction()); //Показывает текущего игрока

GSimCon.directLogAppend("\_BFL\_TEAM\_DOWN\_SIDE");

if (humanPlaterOne.makeTurn() == true) //При успешном выполнении хода

{

gameRule.swapFraction(); //передаём другому игроку ход

GSimCon.getGameSimDataRef()->setCurrFractionTurn(\_BFL\_TEAM\_UP\_SIDE);

}

break;

}

case \_BFL\_TEAM\_UP\_SIDE:

{

GSimCon.getGameSimDataRef()->setCurrFractionTurn(gameRule.getCurrPlayingFraction());

GSimCon.directLogAppend("\_BFL\_TEAM\_UP\_SIDE");

if (humanPlaterTwo.makeTurn() == true)

{

gameRule.swapFraction();

GSimCon.getGameSimDataRef()->setCurrFractionTurn(\_BFL\_TEAM\_DOWN\_SIDE);

}

break;

}

case \_BFL\_TEAM\_NONE:

{

throw \_BFL\_TEAM\_ERR\_NO\_FRACTION\_CHOSEN;

}

default:

break;

}

}

//Инициализация начальных значений игровых элементов

void GameSimulation::GSimCreateStartBaseGameElements(int arrSizeX, int arrSizeY)

{

gameRule.setCurrPlayingFraction(\_BFL\_TEAM\_DOWN\_SIDE);

GSimCon.getGameSimDataRef()->getArrTemplInterface()->createNewArray(arrSizeX, arrSizeY);

GameBaseObject tempObj;

tempObj.setGBOType(\_BFL\_TYPE\_CREATURE\_SQUARE);

tempObj.setGBOName(\_BFL\_STANDART\_NAME);

for (int x = 0; x < arrSizeX; x++)

{

tempObj.setGBOFreeData(std::to\_string(\_BFL\_TEAM\_UP\_SIDE));

GSimCon.getGBOArr()->setElem(tempObj, x, 0);

tempObj.setGBOFreeData(std::to\_string(\_BFL\_TEAM\_DOWN\_SIDE));

GSimCon.getGBOArr()->setElem(tempObj, x, arrSizeY - 1);

}

}

#undef NOT

#undef brk

**GameSimulation.h**

#ifndef \_GAMESIMULATION\_H\_

#define \_GAMESIMULATION\_H\_

#include "OeDataBase.h"

#include "Menu.h"

#include "Player.h"

#include "BattleForLifeGameComponents.h"

#include "GameSimConnector.h"

#include "GameSimStructs.h"

#define GAMESIMULATION\_ERR\_NO\_DATABASE\_PTR "GAMESIMULATION\_ERR\_NO\_DATABASE\_PTR"

/\*

Модуль симуляции игрового процесса

Здесь выполняется логика игры за один прогон цикла

\*/

class GameSimulation;

typedef GameSimulation GameSim;

class GameSimulation

{

private:

GameSimConnector GSimCon; //Коннектор сим.игры с базой данных

BattleForLifeRules gameRule; //Объект с правилами гры

Player humanPlaterOne; //Экземпляры игрока снизу

Player humanPlaterTwo; //Экземпляры игрока сверху

public:

GameSimulation(OEDataBase\* dataBasePtr) :

humanPlaterOne(&gameRule, GSimCon.getDBReference()->getPlayerOneData(), \_BFL\_TEAM\_DOWN\_SIDE),

humanPlaterTwo(&gameRule, GSimCon.getDBReference()->getPlayerTwoData(), \_BFL\_TEAM\_UP\_SIDE),

gameRule(&GSimCon), GSimCon(dataBasePtr)

{

std::cout << "Game sim constr\n";

}

//Инициализация начальных значений сим.игры, накладываемых игровой логикой

void GameSimulationInitialisation();

void setGameSimulationRepherence(OEDataBase\* dataBasePtr) { GSimCon.setDBReference(dataBasePtr); }

bool isGameSim\_NOT\_FullInit()

{

if (NULL == GSimCon.getDBReference() || NULL == GSimCon.getGameSimDataRef())

{

return true;

}

return false;

}

~GameSimulation()

{

GSimCon.~GameSimConnector();

}

void GSimInputHandler(void); //Обработчик входных комманд

void GSimStartMainFunction(void); //главная функция игровой симуляции

void GSimCreateStartBaseGameElements(int arrSizeX, int arrSizeY); //Инициализирует начальные игровые данные

GameSimConnector\* getConnector()

{

return &GSimCon;

}

};

#endif // !\_GAMESIMULATION\_H\_

**GraphicCore.cpp**

#include "GraphicCore.h"

#include "Menu.h"

GrCoreReturnValues GraphicCore::GrCoreStartMainFunc(const GrCorePrintData printData, GrCoreInputArgs arg)

{

system("cls");

GrCoreReturnValues retVal;

retVal.val = 0;

//Связывает все модули и показывает логи происходящего в игре

//std::cout << GrCConnector.getGrCDataRef()->throwFullProcLog() << std::endl;

//Здесь происходит выбор необъодимого типа вывода информации

switch (arg.flag)

{

case \_MENU\_PAGE\_EXIT:

{

std::cout << "\nGame ended\n";

return retVal;

break;

}

case \_MENU\_PAGE\_MAIN:

{

std::cout << GrCConnector.getDBReference()->getMenuData()->getMenuString() << std::endl;

break;

}

case \_MENU\_PAGE\_PAUSE:

{

std::cout << GrCConnector.getDBReference()->getMenuData()->getMenuString() << std::endl;

break;

}

case \_MENU\_PAGE\_GAME\_OVER:

{

std::cout << GrCConnector.getDBReference()->getMenuData()->getMenuString() << std::endl;

break;

}

case(\_MENU\_PAGE\_GREETINGS):

{

std::cout << \_MENU\_GREETING\_MESSAGE;

break;

}

//Главный вывод на экран всех данных из модуля сим.игры

case(\_MENU\_PAGE\_GAME\_SIM):

{

interpretator.convertGameSimDataToArr(); //Конвертируем изменившиеся данные игры

workDataInit(); //подготавливаем промежуточные переменные и объекты

GrCCreateConsoleSTRArrWithSpaces(); //Создание перивод массива игровых объектов в графические объекты

showPossiblePositionInArr(); //Показывает доступные для перемещения клетки поля, при наличии

GrCShowCursors(); //Показывает местоположение курсора

compactSTRArrayToStr(); //Перевод массива строк в обычную строку для вывода на экран

std::cout << \*getConnector()->getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString();

}

default:

break;

}

return retVal;

}

//Функция обработчик входных комманд для граф.ядра

GrCoreReturnValues GraphicCore::GrCoreInputHandler(DBGrCoreData\* p\_DBRawGrCoreData, GrCoreInputArgs args)

{

//на данный момент всегда имеется два действия

GrCoreReturnValues GrCInpHandlret;

GrCInpHandlret.val = 0;

//int\* retVal = &GrCInpHandlret.val;

GrCorePrintData printData = \*p\_DBRawGrCoreData->getGrCConsoleOutDataPtr();

GrCoreReturnValues GrCMainFuncSuccessHandler;

GrCMainFuncSuccessHandler = GrCoreStartMainFunc(printData, args); //запуск главной функции граф.ядра

//Graphic core main function error handler

if (GrCMainFuncSuccessHandler.val != 0) //при неудачном исполнении модуль завершает всю работу

{

std::cout << "Grapric core Dead\n";

GrCInpHandlret.val = -1;

}

return GrCInpHandlret;

}

//Инициализация начальных значений промежуточных компонетов

void GraphicCore::workDataInit()

{

if (workData.\_SPACE\_SIZE <= 0)

{

throw "\_SPACE\_SIZE\_LESS\_ZERO";

}

workData.setGSimArrSize(

interpretator.getMiddleDataRef()->get2DArayInterface()->getSizeHorizonal()

,

interpretator.getMiddleDataRef()->get2DArayInterface()->getSizeVertical()

);

//создание массива строк с дополнительными полями для символов

workData.getLargeGSimStrArr()->createNewArray(

workData.\_SPACE\_SIZE + (workData.\_SPACE\_SIZE + 1) \* workData.getGSimArrSize().getX()

,

workData.\_SPACE\_SIZE + (workData.\_SPACE\_SIZE + 1) \* workData.getGSimArrSize().getY()

);

workData.getLargeGSimStrArr()->fillArrayBy(" ");

}

//Инициализация начальных значений, накладываемых игровой логикой

void GraphicCore::GraphicCoreInitialization(OEDataBase\* p\_DataBase)

{

//GrCConnector.temp = 0;

if (NULL == p\_DataBase)

{

throw GR\_CORE\_ERR\_NO\_DATABASE\_PTR;

}

//инициализация интерпретатора

interpretator.interpretatorInitialisation(p\_DataBase);

std::cout << "GrCore Init\n";

}

//Функция, которая создает массив строк на несколько клеток больше, для дополнительных символов

void GraphicCore::GrCCreateConsoleSTRArrWithSpaces()

{

int consX = workData.getLargeGSimStrArr()->getSizeHorizonal();

int consY = workData.getLargeGSimStrArr()->getSizeVertical();

int maxX, maxY;

maxX = GrCConnector.getConvGSimArrRef()->getSizeHorizonal();

maxY = GrCConnector.getConvGSimArrRef()->getSizeVertical();

int countY = 0;

//Копирует объеты из массива в общих промежуточных данных в "большой" массив с промежутками со сдвигом

for (int y = 0; y < maxY; y++)

{

for (int x = 0; x < maxX; x++)

{

std::string elem = \*GrCConnector.getGrCDataRef()->getMiddleData()->getConvrGSimArr()->getElem(x, y);

int convX = coordFormula(x);

int convY = coordFormula(y);

workData.getLargeGSimStrArr()->setElem(elem, convX, convY);

}

}

}

//Функция переводит строковый массив в одну строку для вывода на экран

void GraphicCore::compactSTRArrayToStr()

{

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->clear();

int consX = workData.getLargeGSimStrArr()->getSizeHorizonal();

int consY = workData.getLargeGSimStrArr()->getSizeVertical();

if (GrCConnector.getDBReference()->getGameSimDataPtr()->getCurrentFractionTurn() == \_BFL\_TEAM\_UP\_SIDE)

{

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append("\t\t\t");

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append(" Up team Turn\n");

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append("\t\t\t");

for (int i = 0; i < consX + 6; i++)

{

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append("V");

}

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append("\n");

}

else

{

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append("\n\n");

}

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append("\t\t\t");

for (int i = 0; i < consX + 6; i++)

{

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append("\_");

}

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append("\n");

for (int y = 0; y < consY; y++)

{

/////////

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append("\t\t\t");

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append("|| ");

////////////

for (int x = 0; x < consX; x++)

{

////////

///////

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append(

\*workData.getLargeGSimStrArr()->getElem(x, y)

);

//////

//////

}

/////////////

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append(" ||");

/////////////

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append("\n");

}

if (GrCConnector.getDBReference()->getGameSimDataPtr()->getCurrentFractionTurn() == \_BFL\_TEAM\_DOWN\_SIDE)

{

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append("\t\t\t");

for (int i = 0; i < consX + 6; i++)

{

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append("^");

}

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append("\n");

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append("\t\t\t");

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append(" Down team Turn\n");

}

else

{

GrCConnector.getGrCDataRef()->getGrCConsoleOutDataPtr()->getOutputString()->append("\n\n\n");

}

return;

}

#define wrkd\_space workData.\_SPACE\_SIZE

//Формула для сдвига координат строкового массива в координаты "большого" массива

int GraphicCore::coordFormula(int coord)

{

return (wrkd\_space + (wrkd\_space + 1) \* coord);

}

vecI2d GraphicCore::convertGSimCoordToLargeArrCoords(vecI2d gSimCoords)

{

if (gSimCoords.getX() >= workData.getGSimArrSize().getX() || gSimCoords.getX() < 0 ||

gSimCoords.getY() >= workData.getGSimArrSize().getY() || gSimCoords.getY() < 0)

{

return \_\_GBO\_VECTOR\_GSIM\_OUT\_OF\_RANGE\_VEC;

}

vecI2d retVec;

int coordFormulaX = coordFormula(gSimCoords.getX());

int coordFormulaY = coordFormula(gSimCoords.getY());

retVec = vecI2d(coordFormulaX, coordFormulaY);

return retVec;

}

//условные "текстуры" курсоров

#define \_UP\_CURSOR\_STR "V"

#define \_DOWN\_CURSOR\_STR "^"

#define \_LEFT\_CURSOR\_STR ">"

#define \_RIGHT\_CURCOR\_STR "<"

#define \_UP\_CURSOR\_PREV\_STR "!"

#define \_DOWN\_CURSOR\_PREV\_STR "A"

#define \_LEFT\_CURSOR\_PREV\_STR "-"

#define \_RIGHT\_CURCOR\_PREV\_STR "-"

#define \_PRINT\_CURSOR\_MODE\_PREV 1

#define \_PRINT\_CURSOR\_MODE\_CURR 2

//Функция по замене элементов "большого" массива на "текстуры" курсоров

void setCursorConvertedCoords(GrCMiddleWorkProcessData\* workData, vecI2d vec, int printCursorMode)

{

switch (printCursorMode)

{

case \_PRINT\_CURSOR\_MODE\_CURR:

{

workData->getLargeGSimStrArr()->setElem(\_UP\_CURSOR\_STR, vec.getX(), vec.getY() - 1);

workData->getLargeGSimStrArr()->setElem(\_DOWN\_CURSOR\_STR, vec.getX(), vec.getY() + 1);

workData->getLargeGSimStrArr()->setElem(\_LEFT\_CURSOR\_STR, vec.getX() - 1, vec.getY());

workData->getLargeGSimStrArr()->setElem(\_RIGHT\_CURCOR\_STR, vec.getX() + 1, vec.getY());

break;

}

case \_PRINT\_CURSOR\_MODE\_PREV:

{

workData->getLargeGSimStrArr()->setElem(\_UP\_CURSOR\_PREV\_STR, vec.getX(), vec.getY() - 1);

workData->getLargeGSimStrArr()->setElem(\_DOWN\_CURSOR\_PREV\_STR, vec.getX(), vec.getY() + 1);

workData->getLargeGSimStrArr()->setElem(\_LEFT\_CURSOR\_PREV\_STR, vec.getX() - 1, vec.getY());

workData->getLargeGSimStrArr()->setElem(\_RIGHT\_CURCOR\_PREV\_STR, vec.getX() + 1, vec.getY());

break;

}

default:

break;

}

}

#define \_PRINT\_POSSIBLE\_POSITION\_SQUARE\_STR "o"

//Функция выводит на экран доступные ходы для клетки для текущего игрока

void GraphicCore::showPossiblePositionInArr()

{

std::vector<vecI2d> possiblePos;

switch (interpretator.getGameSimData()->getCurrentFractionTurn())

{

case \_BFL\_TEAM\_DOWN\_SIDE:

{

possiblePos = GrCConnector.getPlayerOneData()->getPossiblePlayerPos();

break;

}

case \_BFL\_TEAM\_UP\_SIDE:

{

possiblePos = GrCConnector.getPlayerTwoData()->getPossiblePlayerPos();

break;

}

case \_BFL\_TEAM\_NONE:

{

//throw \_BFL\_TEAM\_ERR\_NO\_FRACTION\_CHOSEN;

return;

break;

}

default:

{

return;

break;

}

}

if (possiblePos.empty())

{

return;

}

if (possiblePos[0] == \_BFL\_RULES\_POSSIBLE\_POS\_NOT\_FOUND\_VEC)

{

return;

}

for (int i = 0; i < possiblePos.size(); i++)

{

workData.getLargeGSimStrArr()->setElem(\_PRINT\_POSSIBLE\_POSITION\_SQUARE\_STR,

coordFormula(possiblePos.at(i).getX()),

coordFormula(possiblePos.at(i).getY()));

}

possiblePos.clear();

}

//Фукнция для вывода на экран курсоров текущего игрока

void GraphicCore::GrCShowCursors()

{

switch (interpretator.getGameSimData()->getCurrentFractionTurn())

{

case \_BFL\_TEAM\_DOWN\_SIDE:

{

vecI2d GSimPlayerOneVecCurr = GrCConnector.getPlayerOneData()->getCurrCusor()->getCursorPos();

vecI2d GSimPlayerOneVecFrom = GrCConnector.getPlayerOneData()->getCursorFrom()->getCursorPos();

vecI2d playerOneVecCurr = convertGSimCoordToLargeArrCoords(GSimPlayerOneVecCurr);

vecI2d playerOneVecFrom = convertGSimCoordToLargeArrCoords(GSimPlayerOneVecFrom);

setCursorConvertedCoords(&workData, playerOneVecFrom, \_PRINT\_CURSOR\_MODE\_PREV);

setCursorConvertedCoords(&workData, playerOneVecCurr, \_PRINT\_CURSOR\_MODE\_CURR);

break;

}

case \_BFL\_TEAM\_UP\_SIDE:

{

vecI2d GSimPlayerTwoVecCurr = GrCConnector.getPlayerTwoData()->getCurrCusor()->getCursorPos();

vecI2d GSimPlayerTwoVecFrom = GrCConnector.getPlayerTwoData()->getCursorFrom()->getCursorPos();

vecI2d playerTwoVecCurr = convertGSimCoordToLargeArrCoords(GSimPlayerTwoVecCurr);

vecI2d playerTwoVecFrom = convertGSimCoordToLargeArrCoords(GSimPlayerTwoVecFrom);

setCursorConvertedCoords(&workData, playerTwoVecFrom, \_PRINT\_CURSOR\_MODE\_PREV);

setCursorConvertedCoords(&workData, playerTwoVecCurr, \_PRINT\_CURSOR\_MODE\_CURR);

break;

}

case \_BFL\_TEAM\_NONE:

{

//throw \_BFL\_TEAM\_ERR\_NO\_FRACTION\_CHOSEN;

}

default:

{

break;

}

}

}

#undef wrkd\_space

**GraphicCore.h**

#ifndef GRAPHICCORE\_H\_

#define GRAPHICCORE\_H\_

#include <iostream>

#include "GraphicCoreStructs.h"

#include "OeDataBase.h"

#include "GrCoreGameSimDataInterpretator.h"

#define GR\_CORE\_ERR\_NO\_DATABASE\_PTR "GR\_CORE\_ERR\_NO\_DATABASE\_PTR";

class GrCMiddleWorkProcessData;

/\*

Модуль графического ядра игры

-Здесь реализована логика вывода на экран данных

-Так же здесь происходит выбор каким образом необъодимо показывать данные

\*/

//GraphicCore (OEDataBase\* p\_DataBase)

class GraphicCore

{

public:

GraphicCore(OEDataBase\* p\_DataBase) : GrCConnector(p\_DataBase), interpretator(&GrCConnector)

{

std::cout << "GrCore Constr\n";

}

//Инициализация начальных значений, накладываемых игровой логикой

void GraphicCoreInitialization(OEDataBase\* p\_DataBase);

//обработчких входных значений в графическое ядро

GrCoreReturnValues GrCoreInputHandler(DBGrCoreData\* p\_DBRawGrCoreData, GrCoreInputArgs args);

void compactSTRArrayToStr();

void GrCCreateConsoleSTRArrWithSpaces();

int coordFormula(int coord);

vecI2d convertGSimCoordToLargeArrCoords(vecI2d gSimCoords);

vecI2d convertGSimCoordToLargeArrCoords(int gSimX, int gSimY)

{

return convertGSimCoordToLargeArrCoords(vecI2d(gSimX, gSimY));

}

void GrCShowCursors();

GrCoreDBConnector\* getConnector()

{

return &GrCConnector;

}

GrCMiddleWorkProcessData\* getWrkData()

{

return &workData;

}

void showPossiblePositionInArr();

private:

/\*

Главная функция графического ядра

args:GrCorePrintData, GrCoreInputArgs

\*/

void workDataInit();

GrCoreDBConnector GrCConnector; //Коннектор граф.ядра. Наследник коннектора БД

GrCoreGameSimDataInterpretator interpretator; //Интерпретатор данных игры в графическом виде

GrCMiddleWorkProcessData workData; //данные для удобной работы в пределах модуля

GrCoreReturnValues GrCoreStartMainFunc(GrCorePrintData, GrCoreInputArgs);

};

typedef GraphicCore GrC\_t;

#endif // !GRAPHICCORE\_H\_

**GraphicCoreStructs.h**

#ifndef GRAPHICCORESTRUCTS\_H\_

#define GRAPHICCORESTRUCTS\_H\_

#include <iostream>

#include <string>

#include <sstream>

#include "GameBaseComponents.h"

#include "Vector2DTemplate.h"

#define GRC\_PRINT\_LINE 1

#define GRC\_PRINT\_TO\_LINE\_END 2

#define GRC\_PRINT\_INT\_ARRAY 3

#define GRC\_PRINT\_STR\_ARRAY\_BY\_MAIN\_ONLY 4

typedef std::string StringSTD;

class GrCoreMiddleWorkingData

{

private:

DynamicIntArray2DTemplated<std::string> convertedGameSimArray;

std::string convertedGameSimArrayStr;

DynamicIntArray2DTemplated<std::string> charArray;

public:

DynamicIntArray2DTemplated<std::string>\* get2DArayInterface()

{

return &convertedGameSimArray;

}

DynamicIntArray2DTemplated<std::string>\* getConvrGSimArr()

{

return &convertedGameSimArray;

}

std::string\* getConvertedGSimArrToStr()

{

return &convertedGameSimArrayStr;

}

DynamicIntArray2DTemplated<std::string>\* getCharArr()

{

return &charArray;

}

void setConvertedArray(DynamicIntArray2DTemplated<std::string> newArr)

{

convertedGameSimArray = newArr;

}

GrCoreMiddleWorkingData(){}

};

class GrCorePrintData

{

public:

std::string outputPrintData;

std::string\* getOutputString() { return &consoleOutputStr; }

protected:

//PUT HERE REAL ONJECTS.

//not pointer

std::string consoleOutputStr;

};

class GrCoreGraphicInfo

{

public:

GrCoreGraphicInfo() : consoleOutputStrStream(NULL) {}

std::ostringstream\* getOutStrStreamPtr() { return &consoleOutputStrStream; }

protected:

std::ostringstream consoleOutputStrStream;

};

#define \_GRC\_STRUCTS\_WORKDATA\_ERR\_NO\_LARGE\_ARR "GRC\_STRUCTS\_WORKDATA\_ERR\_NO\_LARGE\_ARR"

/\*

int \_SPACE\_SIZE || 2DArrr<String> LargeArr

vec2D getGSimArrSize || vecI2d getLargeArrMaxSides

\*/

class GrCMiddleWorkProcessData

{

public:

DynamicIntArray2DTemplated<std::string>\* getLargeGSimStrArr()

{

return &largeConsoleGSimStrArr;

}

vecI2d getLargeArrMaxSides()

{

largeArrSizeX = largeConsoleGSimStrArr.getSizeHorizonal();

largeArrSizeY = largeConsoleGSimStrArr.getSizeVertical();

if (largeArrSizeX == 0 || largeArrSizeY == 0)

{

throw \_GRC\_STRUCTS\_WORKDATA\_ERR\_NO\_LARGE\_ARR;

}

return vecI2d(largeArrSizeX, largeArrSizeY);

}

void setGSimArrSize(int sizeX, int sizeY)

{

GSimArrSizeX = sizeX;

GSimArrSizeY = sizeY;

}

VectorInt2D getGSimArrSize()

{

return vecI2d(GSimArrSizeX, GSimArrSizeY);

}

void setFraction(unsigned short int newFraction)

{

fraction = newFraction;

}

unsigned short int getFraction()

{

return fraction;

}

const int \_SPACE\_SIZE = 2;

private:

DynamicIntArray2DTemplated<std::string> largeConsoleGSimStrArr;

unsigned short int fraction;

int GSimArrSizeX;

int GSimArrSizeY;

int largeArrSizeX;

int largeArrSizeY;

};

struct GrCoreInputArgs

{

int flag;

};

/\*

This struct contains output data type and it's params

\*/

struct GrCoreReturnValues

{

int val;

};

//class GrCoreDBConnector;

const std::string TabSpace = "\t";

const std::string Space = " ";

const std::string endll = "\n";

#endif // !GRAPHICCORESTRUCTS\_H\_

**GrCoreGameSimDataInterpretator.cpp**

#include "GrCoreGameSimDataInterpretator.h"

//Инициализация начальных значений интерпретатор, накладываемых игровой логикой

void GrCoreGameSimDataInterpretator::interpretateBaseGameSimInfoToGraphicCoreInfo()

{

//Здесь проверяется сим.игры и создаётся основное поле для работы

if (isNOTGrCInterprFullInitialised())

throw GR\_CORE\_INTERPRET\_ERR\_NOT\_FULL\_INIT;;

int horizontal = getGameSimData()->getArrTemplInterface()->getSizeHorizonal();

int vertical = getGameSimData()->getArrTemplInterface()->getSizeVertical();

getMiddleDataRef()->get2DArayInterface()->createNewArray(horizontal, vertical);

}

//typedef

//Функция перевожа данных игры в данные для вывода на экран

void GrCoreGameSimDataInterpretator::convertGameSimDataToArr()

{

if (isNOTGrCInterprFullInitialised())

{

throw GR\_CORE\_INTERPRET\_ERR\_NOT\_FULL\_INIT;

return;

}

interpretateBaseGameSimInfoToGraphicCoreInfo();

//std::cout << "getSize err in Interpr convert\n";

int horizontal = getGameSimData()->getArrTemplInterface()->getSizeHorizonal();

int vertical = getGameSimData()->getArrTemplInterface()->getSizeVertical();

//Проходим по каждому элементу массива игровых объектов

for (int y = 0; y < vertical; y++)

{

for (int x = 0; x < horizontal; x++)

{

GameBaseObject\* gameSimElem;

gameSimElem = getGameSimData()->getArrTemplInterface()->getElem(x, y);

//Заполнение строкового массива в соответствием с "текстурами" для каждого объекта

if (\_GBC\_OBJECT\_TYPE == \*gameSimElem->getGBOTypePtr())

{

getMiddleDataRef()->get2DArayInterface()->setElem(INTRP\_EMPTY\_GAMEOBJ\_, x, y);

}

else

{

if (gameSimElem->getGBOFreeDataPtr()->\_Equal(\_GBC\_OBJECT\_STANDART\_FREE\_DATA))

{

return;

}

getMiddleDataRef()->get2DArayInterface()->setElem(

\*gameSimElem->getGBOFreeDataPtr(),

x, y);

}

}

}

}

void GrCoreGameSimDataInterpretator::interpretatorInitialisation(OEDataBase\* p\_DataBase)

{

if (NULL == p\_DataBase)

{

throw GR\_CORE\_CONNECTOR\_ERR\_NO\_DATA\_BASE\_PTR;

}

}

**GrCoreGameSimDataInterpretator.h**

//GrCoreGameSimDataInterpretator

#ifndef GRAPHICCORE\_GAMESIMDATA\_INTERPRETATOR\_H\_

#define GRAPHICCORE\_GAMESIMDATA\_INTERPRETATOR\_H\_

#include "OeDataBase.h"

#include "GameSimulation.h"

#define GR\_CORE\_INTERPRET\_ERR\_NO\_CONNECTOR\_PTR "GR\_CORE\_INTERPRET\_ERR\_NO\_CONNECTOR\_PTR"

#define GR\_CORE\_INTERPRET\_ERR\_NOT\_FULL\_INIT "GR\_CORE\_INTERPRET\_ERR\_NOT\_FULL\_INIT"

#define GR\_CORE\_CONNECTOR\_ERR\_NO\_DATA\_BASE\_PTR "GR\_CORE\_CONNECTOR\_ERR\_NO\_DATA\_BASE\_PTR"

/\*

Коннектор графического ядра с базой данных

Хранит в себе прямой доступ к данным граф.ядра

\*/

//GrCoreDBConnector(p\_DataBase)

class GrCoreDBConnector : public OEDBConnector {

public:

GrCoreDBConnector(OEDataBase\* p\_DataBase) : OEDBConnector(p\_DataBase)

{

if (NULL == p\_DataBase)

{

throw GR\_CORE\_CONNECTOR\_ERR\_NO\_DATA\_BASE\_PTR;

}

GrCDataReference = getDBReference()->getGrCData();

convertedGameSimArrayRef = GrCDataReference->getMiddleData()->get2DArayInterface();

charArrayRef = GrCDataReference->getMiddleData()->getCharArr();

}

DBIntelligenceData\* getPlayerOneData()

{

return getDBReference()->getPlayerOneData();

}

DBIntelligenceData\* getPlayerTwoData()

{

return getDBReference()->getPlayerTwoData();

}

DBGrCoreData\* getGrCDataRef()

{

return GrCDataReference;

}

DynamicIntArray2DTemplated<std::string>\* getConvGSimArrRef()

{

return convertedGameSimArrayRef;

}

DynamicIntArray2DTemplated<std::string>\* getCharArrRef()

{

return charArrayRef;

}

protected:

DBGrCoreData\* GrCDataReference;

DynamicIntArray2DTemplated<std::string>\* convertedGameSimArrayRef;

DynamicIntArray2DTemplated<std::string>\* charArrayRef;

};

/\*

Интерпретатор данных игровой симулации

Модуль считывает и переводит все данные игры в удобные для работы

- объекты, которые не зависят от своих оригиналов и не меняют их структуры

\*/

class GrCoreGameSimDataInterpretator;

/\*GrCoreGameSimDataInterpretator(GrCoreDBConnector\* newGrCConRef)

|| (GrCoreDBConnector\* newGrCConRef, GameSimConnector\* newGameSimConRef)\*/

class GrCoreGameSimDataInterpretator

{

public:

const StringSTD INTRP\_EMPTY\_GAMEOBJ\_ = ".";

const StringSTD INTRP\_SMTH\_IN\_GAMEOBJ = "1";

GrCoreGameSimDataInterpretator(GrCoreDBConnector\* newGrCConRef, GameSimConnector\* newGameSimConRef)

{

GrCConReference = newGrCConRef;

middleDataRef = GrCConReference->getGrCDataRef()->getMiddleData();

GameSimConReference = newGameSimConRef;

gameSimDataRef = GameSimConReference->getGameSimDataRef();

}

GrCoreGameSimDataInterpretator(GrCoreDBConnector\* newGrCConRef):

GameSimConReference(NULL), GrCConReference(newGrCConRef),

middleDataRef(GrCConReference->getGrCDataRef()->getMiddleData()),

gameSimDataRef(GrCConReference->getDBReference()->getGameSimDataPtr())

{

std::cout << "Interpr construct start\n";

if (NULL == newGrCConRef)

{

throw GR\_CORE\_INTERPRET\_ERR\_NO\_CONNECTOR\_PTR;

}

std::cout << "Interpr construct end\n";

}

void interpretatorInitialisation(OEDataBase\* p\_DataBase);

~GrCoreGameSimDataInterpretator()

{

GrCConReference = NULL;

}

GrCoreDBConnector\* getGrCConRef()

{

return GrCConReference;

}

GameSimConnector\* getGameSimConRef()

{

return GameSimConReference;

}

bool isNOTGrCInterprFullInitialised()

{

if (NULL == GrCConReference || NULL == gameSimDataRef)

return true;

return false;

}

void interpretateBaseGameSimInfoToGraphicCoreInfo();

void convertGameSimDataToArr();

void setMiddleDataRef(GrCoreMiddleWorkingData\* ref)

{

middleDataRef = ref;

}

GrCoreMiddleWorkingData\* getMiddleDataRef()

{

return middleDataRef;

}

DBGameSimData\* getGameSimData()

{

return gameSimDataRef;

}

private:

GrCoreDBConnector\* GrCConReference;

GrCoreMiddleWorkingData\* middleDataRef;

GameSimConnector\* GameSimConReference;

DBGameSimData\* gameSimDataRef;

};

typedef GrCoreGameSimDataInterpretator GrCInterpr;

#endif GRAPHICCORE\_GAMESIMDATA\_INTERPRETATOR\_H\_// !

**Intelligence.h**

#ifndef \_INTELLIGENCE\_H\_

#define \_INTELLIGENCE\_H\_

#include "OeDataBase.h"

#include "BattleForLifeGameComponents.h"

#define \_BFC\_INTELLIGENCE\_CONNECTOR\_ERR\_NO\_VEC\_FROM "\_BFC\_INTELLIGENCE\_CONNECTOR\_ERR\_NO\_VEC\_FROM"

#define \_BFC\_INTELLIGENCE\_CONNECTOR\_ERR\_NO\_VEC\_TO "\_BFC\_INTELLIGENCE\_CONNECTOR\_ERR\_NO\_VEC\_TO"

#define \_BFL\_INTELLIGENCE\_NO\_ACTION 0

#define \_BFL\_INTELLIGENCE\_CURSOR\_MOVE\_ACTION 1

#define \_BFL\_INTELLIGENCE\_MOVE\_CELL\_CUSTOM 2

#define \_BFL\_INTELLIGENCE\_END\_OF\_TURN 3

#define \_BFL\_INTELLIGENCE\_CREATE\_CQUARE 4

#define \_BFL\_INTELLIGENCE\_RESET 5

#define \_BFL\_INTELLIGENCE\_DYNAMIC\_GBO\_ARR DynamicIntArray2DTemplated<GameBaseObject>

/\*

Коннектор данных интеллекта из БД с самим интеллектом

\*/

class IntelligenceConnector : public OEDBConnector

{

protected:

DBGameSimData\* GSimDataRef;

DBIntelligenceData\* intelDataRef;

public:

IntelligenceConnector(OEDataBase\* p\_OEDataBase, DBIntelligenceData\* intelDatRef) : OEDBConnector(p\_OEDataBase)

{

intelDataRef = intelDatRef;

playerCursorFrom = intelDataRef->getCursorFrom();

playerCursorTo = intelDataRef->getCursorTo();

GSimDataRef = getDBReference()->getGameSimDataPtr();

}

DBIntelligenceData\* getIntelligenceDataRef()

{

return intelDataRef;

}

DBGameSimData\* getGSimDataRef()

{

return GSimDataRef;

}

Cursor\* playerCursorFrom;

Cursor\* playerCursorTo;

~IntelligenceConnector()

{

playerCursorFrom = NULL;

playerCursorTo = NULL;

intelDataRef = NULL;

GSimDataRef = NULL;

}

};

/\*

Базовый класс интеллекта игрока

Здесь содержатся общие и обязательные для всех игроков команды

\*/

class Intelligence

{

protected:

\_FRACTION\_INT\_ playerFraction; //Текущая команда

IntelligenceConnector intelCon; //Коннектор

int chosenAction = \_BFL\_INTELLIGENCE\_NO\_ACTION; //Действие интеллекта

BattleForLifeRules\* BFLRules; //Ссылка на правила игры

public:

\_FRACTION\_INT\_ getFraction()

{

return playerFraction;

}

\_BFL\_INTELLIGENCE\_DYNAMIC\_GBO\_ARR\* getArr()

{

return intelCon.getGSimDataRef()->getArrTemplInterface();

}

Intelligence(OEDataBase\* p\_OEDataBase, DBIntelligenceData\* dataRef, \_FRACTION\_INT\_ fraction, BattleForLifeRules\* rules) :

playerFraction(fraction), intelCon(p\_OEDataBase, dataRef),

BFLRules(rules) {}

Cursor\* getCursorFrom()

{

if (NULL == intelCon.playerCursorFrom)

{

throw \_BFC\_INTELLIGENCE\_CONNECTOR\_ERR\_NO\_VEC\_FROM;

}

return intelCon.playerCursorFrom;

}

Cursor\* getCursorTo()

{

if (NULL == intelCon.playerCursorTo)

{

throw \_BFC\_INTELLIGENCE\_CONNECTOR\_ERR\_NO\_VEC\_TO;

}

return intelCon.playerCursorTo;

}

Cursor\* getCurrCursor()

{

return intelCon.getIntelligenceDataRef()->getCurrCusor();

}

GameBaseObject\* getElemByCursor(Cursor\* cursor)

{

GameBaseObject\* retObj;

retObj = intelCon.getGSimDataRef()->getArrTemplInterface()->getElem(

cursor->getCursorPos().getX(),

cursor->getCursorPos().getY()

);

return retObj;

}

BattleForLifeRules\* getRules()

{

return BFLRules;

}

void setAction(int action)

{

chosenAction = action;

}

//обязательная для выполения функция выдачи команды

int giveAction()

{

int temp = chosenAction;

chosenAction = \_BFL\_INTELLIGENCE\_NO\_ACTION;

return temp;

}

virtual ~Intelligence()

{

BFLRules = NULL;

}

virtual bool isSquareInMyTeam(int x, int y)

{

GameBaseObject curObject;

curObject = \*intelCon.getGSimDataRef()->getArrTemplInterface()->getElem(x, y);

if (curObject.getGBOTypePtr()->\_Equal(\_GBC\_OBJECT\_TYPE))

{

return true;

}

if (!curObject.getGBOFreeDataPtr()->\_Equal(std::to\_string(playerFraction)))

{

return false;

}

return true;

}

//обязательная для исполнения функция инициализации

virtual void intelligenceInit() {};

//обязательная для исполнения функция выбора действия

virtual void chooseAction() {};

};

#endif // !\_INTELLIGENCE\_H\_

**Menu.cpp**

#include "Menu.h"

#include <iostream>

#include <windows.h>

#include <conio.h>

#include <string>

#include "BattleForLifeGameComponents.h"

#define brk break

void MenuInterface::keyBoardInput()

{

if (menuCon.getMenuData()->getNeedToSkipInputStatus() == true)

{

menuCon.getMenuData()->setNeedToSkipInput(false);

menuCon.getMenuData()->setKeyCase(\_MENU\_CASE\_NO\_ARG);

return;

}

std::string inpStr = std::to\_string(\_MENU\_CASE\_START\_VALUE);

ULONGLONG toGo = 1000, start = GetTickCount64();

int keyInt = \_MENU\_CASE\_START\_VALUE;

while ((start + toGo) >= GetTickCount64())

{

if (\_kbhit())

{

keyInt = \_getch();

menuCon.directLogAppend(std::to\_string(keyInt));

break;

}

else

{

continue;

}

}

menuCon.getMenuData()->setKeyCase(useMenuSwitch(keyInt));

}

void MenuInterface::solveCommand()

{

int curPage = menuCon.getMenuData()->getPage();

switch (curPage)

{

case \_MENU\_PAGE\_EXIT:

{

brk;

}

case \_MENU\_PAGE\_MAIN:

{

mainMenuPage();

brk;

}

case \_MENU\_PAGE\_GAME\_SIM:

{

brk;

}

case \_MENU\_PAGE\_GAME\_OVER:

{

gameOverPage();

brk;

}

case \_MENU\_PAGE\_PAUSE:

{

pausePage();

break;

}

case \_MENU\_PAGE\_RESTART\_GAME:

{

brk;

}

default:

break;

}

}

bool MenuInterface::isCaseExists(int keyInt)

{

if (\_MENU\_CASE\_NO\_ARG == keyInt)

{

return false;

}

return true;

}

#define \_MENU\_KEY\_CURS\_UP 38

#define \_MENU\_KEY\_CURS\_DOWN 40

#define \_MENU\_KEY\_CURS\_LEFT 37

#define \_MENU\_KEY\_CURS\_RIGHT 39

#define \_MENU\_KEY\_W 119

#define \_MENU\_KEY\_A 97

#define \_MENU\_KEY\_S 115

#define \_MENU\_KEY\_D 100

#define \_MENU\_KEY\_E 101

#define \_MENU\_KEY\_ESC 27

#define \_MENU\_KEY\_ENTER 13

#define \_MENU\_KEY\_BACKSPACE 8

int MenuInterface::useMenuSwitch(int keyCommand)

{

switch (keyCommand)

{

case \_MENU\_KEY\_CURS\_UP:{}

case \_MENU\_KEY\_W:

{

return \_MENU\_CASE\_SET\_UP;

brk;

}

case \_MENU\_KEY\_S:{}

case \_MENU\_KEY\_CURS\_DOWN:

{

return \_MENU\_CASE\_SET\_DOWN;

brk;

}

case \_MENU\_KEY\_A:{}

case \_MENU\_KEY\_CURS\_LEFT:

{

return \_MENU\_CASE\_SET\_LEFT;

brk;

}

case \_MENU\_KEY\_CURS\_RIGHT:{}

case \_MENU\_KEY\_D:

{

return \_MENU\_CASE\_SET\_RIGHT;

brk;

}

case \_MENU\_KEY\_ESC:

{

return \_MENU\_CASE\_EXIT;

brk;

}

case \_MENU\_KEY\_E: {}

case \_MENU\_KEY\_ENTER:

{

return \_MENU\_CASE\_SET\_ENTER;

brk;

}

case \_MENU\_KEY\_BACKSPACE:

{

return \_MENU\_CASE\_SET\_RESET;

brk;

}

default:

{

return \_MENU\_CASE\_NO\_ARG;

break;

}

}

}

#define \_MENU\_MAIN\_PAGE\_WIDTH 1

#define \_MENU\_MAIN\_PAGE\_HEIGHT 2

std::string MenuInterface::pausePage()

{

std::vector<std::string> strArr;

std::string retStr;

const std::string resumeGame = "Resume game";

const std::string exitGame = "Exit game";

const std::string backToMainMenu = "Return to main menu";

strArr.push\_back(resumeGame);

strArr.push\_back(backToMainMenu);

strArr.push\_back(exitGame);

retStr.append("\n\n");

Cursor\* crsRef = menuCon.getMenuData()->getCursor();

if (!crsRef->isLimitsSet() || menuCon.getMenuData()->getPausePageStatus() == false)

{

menuCon.getMenuData()->setPausePageStatus(true);

crsRef->setBotRightLim(vecI2d(\_MENU\_MAIN\_PAGE\_WIDTH, strArr.size()));

crsRef->setTopLeftLim(vecI2d(0, 0));

crsRef->setCursor(vecI2d(0, 0));

}

menuCon.directLogAppend(crsRef->getCursorPos().getStrCoords());

int menuCase = menuCon.getMenuData()->getKeyCase();

menuCon.directLogAppend(std::to\_string(menuCase));

switch (menuCase)

{

case \_MENU\_CASE\_SET\_UP:

{

crsRef->crsUP();

brk;

}

case \_MENU\_CASE\_SET\_DOWN:

{

crsRef->crsDOWN();

brk;

}

case \_MENU\_CASE\_SET\_ENTER:

{

int curY = crsRef->getCursorPos().getY();

std::string strArrElem = strArr.at(curY);

if (strArrElem.\_Equal(resumeGame))

{

menuCon.getMenuData()->setPage(\_MENU\_PAGE\_GAME\_SIM);

menuCon.getMenuData()->setPausePageStatus(false);

}

if (strArrElem.\_Equal(exitGame))

{

menuCon.getMenuData()->setPage(\_MENU\_PAGE\_EXIT);

menuCon.getMenuData()->setPausePageStatus(false);

}

if (strArrElem.\_Equal(backToMainMenu))

{

menuCon.getMenuData()->setPage(\_MENU\_PAGE\_MAIN);

menuCon.getMenuData()->setPausePageStatus(false);

}

brk;

}

default:

menuCon.getMenuData()->setKeyCase(\_MENU\_CASE\_NO\_ARG);

break;

}

for (int y = 0; y < strArr.size(); y++)

{

if (crsRef->getCursorPos().getY() == y)

{

retStr.append(">>>>\t");

}

else

{

retStr.append("\t");

}

retStr.append(strArr.at(y));

retStr.append("\n");

}

retStr.append("\n\tGame by Moskalev Yuri Antonovich. \n\n");

retStr.append("\tNizhniy Novgorod. NGTU. 2022\n\n");

menuCon.getMenuData()->setMenuString(retStr);

return retStr;

}

#define \_MENU\_PAUSE\_PAGE\_WIDTH 1

#define \_MENU\_PAUSE\_PAGE\_HEIGTH 1

std::string MenuInterface::mainMenuPage()

{

std::vector<std::string> strArr;

std::string retStr;

const std::string exitGame = "Exit game";

const std::string playWithBot = "Person VS Robot";

const std::string playerWithPlayer = "Person VS Person";

const std::string botVSbot = "Robot VS Robot";

const std::string changeArrSize = "Change Game Sim array size";

strArr.push\_back(playWithBot);

strArr.push\_back(botVSbot);

strArr.push\_back(playerWithPlayer);

strArr.push\_back(changeArrSize);

strArr.push\_back(exitGame);

Cursor\* crsRef = menuCon.getMenuData()->getCursor();

if (!crsRef->isLimitsSet() || menuCon.getMenuData()->getMainMenuStatus() == false)

{

menuCon.getMenuData()->setMainMenuActive(true);

crsRef->setBotRightLim(vecI2d(\_MENU\_MAIN\_PAGE\_WIDTH, strArr.size()));

crsRef->setTopLeftLim(vecI2d(0, 0));

crsRef->setCursor(vecI2d(0, 0));

}

retStr.append("\n");

retStr.append("\tBattle for life\n\n");

retStr.append("\t-WASD- for navigation. ENTER button for selection command\n\n");

menuCon.directLogAppend(crsRef->getCursorPos().getStrCoords());

int menuCase = menuCon.getMenuData()->getKeyCase();

menuCon.directLogAppend(std::to\_string(menuCase));

switch (menuCase)

{

case \_MENU\_CASE\_SET\_UP:

{

crsRef->crsUP();

brk;

}

case \_MENU\_CASE\_SET\_DOWN:

{

crsRef->crsDOWN();

brk;

}

case \_MENU\_CASE\_SET\_ENTER:

{

int curY = crsRef->getCursorPos().getY();

std::string strArrElem = strArr.at(curY);

if (strArrElem.\_Equal(exitGame))

{

menuCon.getMenuData()->setPage(\_MENU\_PAGE\_EXIT);

menuCon.getMenuData()->setMainMenuActive(false);

}

if (strArrElem.\_Equal(playWithBot))

{

menuCon.getMenuData()->setPage(\_MENU\_PAGE\_PLAY\_WITH\_BOT);

menuCon.getMenuData()->setMainMenuActive(false);

}

if (strArrElem.\_Equal(botVSbot))

{

menuCon.getMenuData()->setPage(\_MENU\_PAGE\_PLAY\_BY\_BOTS);

menuCon.getMenuData()->setMainMenuActive(false);

}

if (strArrElem.\_Equal(playerWithPlayer))

{

menuCon.getMenuData()->setPage(\_MENU\_PAGE\_PLAY\_PERSON\_WITH\_PERSON);

menuCon.getMenuData()->setMainMenuActive(false);

}

if (strArrElem.\_Equal(changeArrSize))

{

std::cout << "\nInsert horizontal size.\nMinimum 3. Max 7: ";

int sizeH = -1, sizeV = -1;

std::cin >> sizeH;

if (sizeH > 7 || sizeH < 3)

{

std::cout << "\nUnvalid arr size. Try Again";

Sleep(1000);

return "Unvalid";

}

std::cout << "\nInsert vertical size.\nMinimum 3. Max 7: ";

std::cin >> sizeV;

if (sizeV > 7 || sizeV < 3)

{

std::cout << "\nUnvalid arr size. Try Again";

Sleep(1000);

return "Unvalid";

}

menuCon.getDBReference()->getGameSimDataPtr()->setGSimArrSizeHorizontal(sizeH);

menuCon.getDBReference()->getGameSimDataPtr()->setGSimArrSizeVertical(sizeV);

menuCon.getMenuData()->setPage(\_MENU\_PAGE\_CHANGE\_ARR\_SIZE);

menuCon.getMenuData()->setMainMenuActive(false);

menuCon.getMenuData()->setNeedToSkipInput(true);

}

brk;

}

default:

menuCon.getMenuData()->setKeyCase(\_MENU\_CASE\_NO\_ARG);

break;

}

for (int y = 0; y < strArr.size(); y++)

{

if (crsRef->getCursorPos().getY() == y)

{

retStr.append(">>>>\t");

}

else

{

retStr.append("\t");

}

retStr.append(strArr.at(y));

if (strArr.at(y).\_Equal(changeArrSize)

&& menuCon.getDBReference()->getGameSimDataPtr()->getGSimArrSizeHorizontal() != \_GSIM\_ARR\_SIZE\_NOT\_SET

&& menuCon.getDBReference()->getGameSimDataPtr()->getGSimArrSizeVertical() != \_GSIM\_ARR\_SIZE\_NOT\_SET

)

{

retStr.append(" GSim size("

+ std::to\_string(menuCon.getDBReference()->getGameSimDataPtr()->getGSimArrSizeHorizontal())

+ " , "

+ std::to\_string(menuCon.getDBReference()->getGameSimDataPtr()->getGSimArrSizeVertical())

+ ")"

);

}

retStr.append("\n");

}

retStr.append("\n\tGame by Moskalev Yuri Antonovich. \n\n");

retStr.append("\tNizhniy Novgorod. NGTU. 2022\n\n");

menuCon.getMenuData()->setMenuString(retStr);

return retStr;

}

#define \_MENU\_GAME\_OVER\_PAGE\_WIDTH 1

#define \_MENU\_GAME\_OVER\_PAGE\_HEIGHT 2

std::string MenuInterface::gameOverPage()

{

std::vector<std::string> strArr;

std::string retStr;

const std::string restartGame = "Restart game";

const std::string exitGame = "Exit game";

const std::string backToMainMenu = "Return to main menu";

strArr.push\_back(restartGame);

strArr.push\_back(backToMainMenu);

strArr.push\_back(exitGame);

Cursor\* crsRef = menuCon.getMenuData()->getCursor();

if (!crsRef->isLimitsSet() || menuCon.getMenuData()->getGameOverPageStatus() == false)

{

menuCon.getMenuData()->setGameOverMenu(true);

crsRef->setBotRightLim(vecI2d(\_MENU\_GAME\_OVER\_PAGE\_WIDTH, strArr.size()));

crsRef->setTopLeftLim(vecI2d(0, 0));

crsRef->setCursor(vecI2d(0, 0));

}

retStr.append("\n\n");

menuCon.directLogAppend(crsRef->getCursorPos().getStrCoords());

int menuCase = menuCon.getMenuData()->getKeyCase();

menuCon.directLogAppend(std::to\_string(menuCase));

switch (menuCase)

{

case \_MENU\_CASE\_SET\_UP:

{

crsRef->crsUP();

brk;

}

case \_MENU\_CASE\_SET\_DOWN:

{

crsRef->crsDOWN();

brk;

}

case \_MENU\_CASE\_SET\_ENTER:

{

int curY = crsRef->getCursorPos().getY();

std::string strArrElem = strArr.at(curY);

if (strArrElem.\_Equal(restartGame))

{

menuCon.getMenuData()->setPage(\_MENU\_PAGE\_RESTART\_GAME);

menuCon.getMenuData()->setPausePageStatus(false);

menuCon.getMenuData()->setNeedToSkipInput(true);

}

if (strArrElem.\_Equal(exitGame))

{

menuCon.getMenuData()->setPage(\_MENU\_PAGE\_EXIT);

menuCon.getMenuData()->setPausePageStatus(false);

}

if (strArrElem.\_Equal(backToMainMenu))

{

menuCon.getMenuData()->setPage(\_MENU\_PAGE\_MAIN);

menuCon.getMenuData()->setPausePageStatus(false);

menuCon.getMenuData()->setNeedToSkipInput(true);

}

brk;

}

default:

menuCon.getMenuData()->setKeyCase(\_MENU\_CASE\_NO\_ARG);

break;

}

if (menuCon.getDBReference()->getMenuData()->getLosedFraction() == \_BFL\_TEAM\_UP\_SIDE)

{

retStr.append("\tThe Up team lose the game\n\n");

}

else if (menuCon.getDBReference()->getMenuData()->getLosedFraction() == \_BFL\_TEAM\_DOWN\_SIDE)

{

retStr.append("\tThe Down team lose the game\n\n");

}

else if(menuCon.getDBReference()->getMenuData()->getLosedFraction() == \_BFL\_TEAM\_NONE)

{

retStr.append("\tFriendship won the game\n\n");

}

for (int y = 0; y < strArr.size(); y++)

{

if (crsRef->getCursorPos().getY() == y)

{

retStr.append(">>>>\t");

}

else

{

retStr.append("\t");

}

retStr.append(strArr.at(y));

retStr.append("\n");

}

retStr.append("\n\tGame by Moskalev Yuri Antonovich. \n\n");

retStr.append("\tNizhniy Novgorod. NGTU. 2022\n\n");

menuCon.getMenuData()->setMenuString(retStr);

return retStr;

}

#undef brk

**Menu.h**

#ifndef MENU\_H\_

#define MENU\_H\_

#include "OeDataBase.h"

#include "OEDataBaseData.h"

#define \_MENU\_CASE\_NO\_ARG -1

#define \_MENU\_PAGE\_EXIT 0

#define \_MENU\_PAGE\_GREETINGS 1

#define \_MENU\_PAGE\_MAIN 2

#define \_MENU\_PAGE\_GAME\_SIM 3

#define \_MENU\_PAGE\_PAUSE 4

#define \_MENU\_PAGE\_GAME\_OVER 5

#define \_MENU\_PAGE\_RESTART\_GAME 6

#define \_MENU\_CASE\_START\_VALUE -999

#define \_MENU\_CASE\_EXIT 0

#define \_MENU\_CASE\_SET\_UP 1

#define \_MENU\_CASE\_SET\_DOWN 2

#define \_MENU\_CASE\_SET\_LEFT 3

#define \_MENU\_CASE\_SET\_RIGHT 4

#define \_MENU\_CASE\_SET\_ENTER 5

#define \_MENU\_CASE\_SET\_RESET 6

#define \_MENU\_PAGE\_PLAY\_WITH\_BOT 7

#define \_MENU\_PAGE\_PLAY\_BY\_BOTS 8

#define \_MENU\_PAGE\_PLAY\_PERSON\_WITH\_PERSON 9

#define \_MENU\_PAGE\_CHANGE\_ARR\_SIZE 10

class MenuConnector : public OEDBConnector

{

private:

DBMenuData\* menuDataRef;

public:

MenuConnector(OEDataBase\* p\_dataBase) : OEDBConnector(p\_dataBase)

{

menuDataRef = getDBReference()->getMenuData();

}

DBMenuData\* getMenuData()

{

return menuDataRef;

}

};

class MenuInterface

{

private:

MenuConnector menuCon;

public:

MenuInterface(OEDataBase\* p\_dataBase) : menuCon(p\_dataBase){}

void keyBoardInput();

std::string pausePage();

void solveCommand();

bool isCaseExists(int keyInt);

int useMenuSwitch(int keyCommand);

std::string mainMenuPage();

std::string gameOverPage();

};

//"BATTLE FOR LIFE"

const std::string \_MENU\_GREETING\_MESSAGE("\n\n\t\

### ## ##### ##### # ##### ##### #### #### \n\t\

# # # # # # # # # # # # # \n\t\

### # # # # # #### ### # # #### \n\t\

# # #### # # # # # # # # # \n\t\

### # # # # ##### ##### # #### # # \n\

\n\t\t\t\

# ##### ##### ##### \n\t\t\t\

# # # # \n\t\t\t\

# # ### ### \n\t\t\t\

# # # # \n\t\t\t\

##### ##### # ##### \n\

");

#endif // !MENU\_H\_

**OeDataBase.h**

#ifndef OEDATABASE\_H\_

#define OEDATABASE\_H\_

/\*

Class OEDataBase:

Содержит поля:

DBGrCoredata GrCData

..............

\*/

#include "OeDataBasePrototypes.h"

#include "GraphicCoreStructs.h"

#include "GameBaseComponents.h"

#include "OEDataBaseData.h"

class OEDBConnector;

#define OEDATABASE\_CONNECTOR\_ERR\_NO\_DB\_PTR "OEDATABASE\_CONNECTOR\_ERR\_NO\_DATABASE\_PTR"

/\*

База данных со всеми важными переменными и объектами,

у которых есть необходимость перемещаться между модулями

Это позволяет избегать дополнительного копирования данных

\*/

class OEDataBase

{

public:

OEDataBase()

{

std::cout << "Data base constr \n";

}

~OEDataBase(){}

DBGameSimData\* getGameSimDataPtr()

{

return &GameSimData;

}

DBGrCoreData\* getGrCData()

{

return &GrCData;

}

DBMenuData\* getMenuData()

{

return &MenuData;

}

DBIntelligenceData\* getPlayerOneData()

{

return GameSimData.getPlayerOneData();

}

DBIntelligenceData\* getPlayerTwoData()

{

return GameSimData.getPlayerTwoData();

}

private:

DBGrCoreData GrCData; //данные граф.ядра

DBGameSimData GameSimData; //данные ядра сим.игры

DBMenuData MenuData; //данные с меню

};

/\*

Коннектор - это интерфейс доступа к базе данных и её содержимому

У каждого модуля могут быть свои коннекторы, но они все должны соблюдать

- общую логику доступа к базе.

Здесь базовый класс для коннектора базы данных

\*/

class OEDBConnector{

public:

OEDataBase\* getDBReference(){ return DBReference; }

void setDBReference(OEDataBase\* DataBaseReference)

{

DBReference = DataBaseReference;

}

//OEDBConnector() {}

OEDBConnector( OEDataBase\* p\_DataBase)

{

if (NULL == p\_DataBase)

{

throw OEDATABASE\_CONNECTOR\_ERR\_NO\_DB\_PTR;

}

DBReference = p\_DataBase;

}

void directLogAppend(std::string log)

{

getDBReference()->getGrCData()->appendProcLog(log);

}

~OEDBConnector() { DBReference = NULL; }

protected:

OEDataBase\* DBReference;

};

#endif // OEDATABASE\_H\_

**OEDataBaseData.h**

#ifndef \_OEDATABASE\_DATA\_

#define \_OEDATABASE\_DATA\_

#include "GraphicCoreStructs.h"

#include "Vector2DTemplate.h"

#include "GameBaseComponents.h"

#include "GameSimStructs.h"

class DBGrCoreData;

class DBGrCoreData

{

public:

void setGrCConsoleOutData(GrCorePrintData newGrCConsoleOut) { ConsoleOut = newGrCConsoleOut; }

GrCorePrintData\* getGrCConsoleOutDataPtr() { return &ConsoleOut; }

GrCoreMiddleWorkingData\* getMiddleData() { return &middleData; }

DBGrCoreData()

{

ConsoleOut = GrCorePrintData();

}

~DBGrCoreData(){}

void appendProcLog(std::string newMessage)

{

processLog.append(newMessage + "\n");

}

std::string throwFullProcLog()

{

std::string retStr = processLog;

processLog.clear();

return retStr;

}

private:

//PUT HERE REAL ONJECTS.

//not pointer

GrCorePrintData ConsoleOut;

std::string processLog;

GrCoreMiddleWorkingData middleData;

};

#define \_FROM\_CURSOR 0

#define \_TO\_CURSOR 1

#define \_UNS\_SHORT\_INT unsigned short int

#define \_DB\_INTELLIGENCE\_DATA\_NO\_SWAP\_CURSORES -1

class DBIntelligenceData

{

protected:

Cursor playerCursorFrom;

Cursor playerCursorTo;

\_UNS\_SHORT\_INT currCursor = \_FROM\_CURSOR;

std::vector<vecI2d> possiblePlayerPos;

public:

std::vector<vecI2d> getPossiblePlayerPos()

{

return possiblePlayerPos;

}

std::vector<vecI2d> givePossiblePlayerPos()

{

std::vector<vecI2d> temp = possiblePlayerPos;

possiblePlayerPos.clear();

return temp;

}

void clearPossiblePos()

{

possiblePlayerPos.clear();

}

void setPossiblePlayerPos(std::vector<vecI2d> curPossiblePos)

{

possiblePlayerPos.clear();

possiblePlayerPos = curPossiblePos;

}

DBIntelligenceData() {}

\_UNS\_SHORT\_INT getCurrCursorNum()

{

return currCursor;

}

Cursor\* getCurrCusor()

{

switch (getCurrCursorNum())

{

case \_FROM\_CURSOR:

{

return &playerCursorFrom;

break;

}

case \_TO\_CURSOR:

{

return &playerCursorTo;

break;

}

default:

break;

}

return NULL;

}

\_UNS\_SHORT\_INT swapCursor()

{

if (currCursor == \_FROM\_CURSOR)

{

currCursor = \_TO\_CURSOR;

return currCursor;

}

else if (currCursor == \_TO\_CURSOR)

{

currCursor = \_FROM\_CURSOR;

return currCursor;

}

return \_DB\_INTELLIGENCE\_DATA\_NO\_SWAP\_CURSORES;

}

Cursor\* getCursorFrom()

{

return &playerCursorFrom;

}

Cursor\* getCursorTo()

{

return &playerCursorTo;

}

void setCursorFrom(VectorInt2D cursorFrom)

{

playerCursorFrom.setCursor(cursorFrom.getX(), cursorFrom.getY());

}

void setCursorTo(VectorInt2D cursorTo)

{

playerCursorTo.setCursor(cursorTo.getX(), cursorTo.getY());

}

};

class DBGameSimData;

class DBGameSimData

{

public:

DBGameSimData()

{

std::cout << "DBGameSimData constr" << std::endl;

}

~DBGameSimData() {}

DynamicIntArray2DTemplated<GameBaseObject>\* getArrTemplInterface()

{

return &gameSimGBOArr2DTempl;

}

void setGameSimMenuMode(int newMode)

{

gameSimMenuInputMode = newMode;

}

int getGameSimMenuMode()

{

return gameSimMenuInputMode;

}

void setCurrFractionTurn(unsigned short int fraction)

{

currentFractionTurn = fraction;

}

unsigned short int getCurrentFractionTurn()

{

return currentFractionTurn;

}

DBIntelligenceData\* getPlayerOneData()

{

return &PlayerOneData;

}

DBIntelligenceData\* getPlayerTwoData()

{

return &PlayerTwoData;

}

void setGSimArrSizeHorizontal(int newSize)

{

GSimArrSizeHor = newSize;

}

int getGSimArrSizeHorizontal()

{

return GSimArrSizeHor;

}

void setGSimArrSizeVertical(int newSize)

{

GSimArrSizeVert = newSize;

}

int getGSimArrSizeVertical()

{

return GSimArrSizeVert;

}

private:

DynamicIntArray2DTemplated<GameBaseObject> gameSimGBOArr2DTempl;

unsigned short int currentFractionTurn;

int gameSimMenuInputMode = 0;

int GSimArrSizeHor = \_GSIM\_ARR\_SIZE\_NOT\_SET;

int GSimArrSizeVert = \_GSIM\_ARR\_SIZE\_NOT\_SET;

DBIntelligenceData PlayerOneData;

DBIntelligenceData PlayerTwoData;

};

class DBMenuData

{

private:

std::string menuString;

int keyInputInt;

int keyCase;

bool isNeedToSkipInput = false;

bool isMainMenuActive = true;

bool isPausePageActive = false;

bool isGameOverPageActive = false;

unsigned short int losedFraction = 0;

int menuPage;

Cursor menuCursor;

public:

DBMenuData(){}

void setLosedFraction(unsigned short int team)

{

losedFraction = team;

}

unsigned short int getLosedFraction()

{

return losedFraction;

}

void setNeedToSkipInput(bool onOff)

{

isNeedToSkipInput = onOff;

}

bool getNeedToSkipInputStatus()

{

return isNeedToSkipInput;

}

void setGameOverMenu(bool onOff)

{

isGameOverPageActive = onOff;

}

bool getGameOverPageStatus()

{

return isGameOverPageActive;

}

void setMenuString(std::string newStr)

{

menuString = newStr;

}

void setMainMenuActive(bool onOff)

{

isMainMenuActive = onOff;

}

bool getMainMenuStatus()

{

return isMainMenuActive;

}

void setPausePageStatus(bool onOff)

{

isPausePageActive = onOff;

}

bool getPausePageStatus()

{

return isPausePageActive;

}

std::string getMenuString()

{

return menuString;

}

void setPage(int newPage)

{

setNeedToSkipInput(true);

menuPage = newPage;

}

int getPage()

{

return menuPage;

}

Cursor\* getCursor()

{

return &menuCursor;

}

void setKeyCase(int cse)

{

keyCase = cse;

}

int getKeyCase()

{

return keyCase;

}

int\* getKeyInputInt()

{

return &keyInputInt;

}

void setKeyInputInt(int intInput)

{

keyInputInt = intInput;

}

};

#endif // !\_OEDATABASE\_DATA\_

**PersonIntelligence.cpp**

#include "PersonIntelligence.h"

#include "GameBaseComponents.h"

#include "GameSimStructs.h"

#define brk break

void PersonIntelligence::intelligenceInit()

{

int x = intelCon.getGSimDataRef()->getArrTemplInterface()->getSizeHorizonal();

int y = intelCon.getGSimDataRef()->getArrTemplInterface()->getSizeVertical();

if (x <= 0 || y <= 0)

{

throw "NO\_WALUES";

}

getCursorFrom()->setTopLeftLim(vecI2d(0, 0));

getCursorFrom()->setBotRightLim(vecI2d(x, y));

getCursorFrom()->setCursor(x / 2, y / 2);

getCursorTo()->setTopLeftLim(vecI2d(0, 0));

getCursorTo()->setBotRightLim(vecI2d(x, y));

getCursorTo()->setCursor(x / 2, y / 2);

}

//Действие при нажатии enter

void PersonIntelligence::menuComEnterFromCursorAction()

{

//Если клетка из моей команды

if (isSquareInMyTeam(getCursorFrom()->getCursorPos().getX(),

getCursorFrom()->getCursorPos().getY()

))

{

GameBaseObject curObject;

curObject = \*intelCon.getGSimDataRef()->getArrTemplInterface()->getElem(

getCursorFrom()->getCursorPos().getX(),

getCursorFrom()->getCursorPos().getY()

);

if (!curObject.getGBOTypePtr()->\_Equal(\_GBC\_OBJECT\_TYPE)) //Если не пустая клетка

{

std::vector<vecI2d> possiblePosArr;

intelCon.getIntelligenceDataRef()->clearPossiblePos(); //очищаем предыдущие доступные ходы

possiblePosArr = getRules()->showAllPosiblePositions(getCursorFrom()->getCursorPos()); //показываем доступыне ходы

intelCon.getIntelligenceDataRef()->setPossiblePlayerPos(possiblePosArr); //отправляем в БД информацию о доступных полях

}

intelCon.getIntelligenceDataRef()->swapCursor(); //меняем курсор

intelCon.getIntelligenceDataRef()->getCursorTo()->setCursor(

intelCon.getIntelligenceDataRef()->getCursorFrom()->getCursorPos()

); //двигаем положение курсора конечной точки в положение курсора изначальной точки

}

}

//комманда для сброса текущего действия, если у него несколько шагов

void PersonIntelligence::menuCommandReset()

{

if (intelCon.getIntelligenceDataRef()->getCurrCursorNum() == \_TO\_CURSOR) //если сейчас курсор конечной точки

{

intelCon.getIntelligenceDataRef()->getCurrCusor()->setCursor

(

intelCon.getIntelligenceDataRef()->getCursorFrom()->getCursorPos()

); //перемещаем курсор конечной точки, в точку, откуда мы хотели совершить действие ранее

intelCon.getIntelligenceDataRef()->swapCursor(); //меняем курсор конечной точки на курсор началбной точки

}

intelCon.getIntelligenceDataRef()->clearPossiblePos(); //убираем все доступные действия для клетки

}

//Обратка вариантов действий при нажатие на enter

void PersonIntelligence::menuCommandEnter()

{

if (intelCon.getIntelligenceDataRef()->getCurrCursorNum() == \_FROM\_CURSOR) //если сейчас курсор начальной точки

{

menuComEnterFromCursorAction(); //Выполняем действие для курсора в первый раз

return;

}

else if (intelCon.getIntelligenceDataRef()->getCurrCursorNum() == \_TO\_CURSOR //если текущий курсор - это конечная точка

&& getElemByCursor(getCurrCursor())->getGBOTypePtr()->\_Equal(\_GBC\_OBJECT\_TYPE))

{

if (intelCon.getIntelligenceDataRef()->getCursorFrom()->getCursorPos() //если положения двух курсоров совпадают

== intelCon.getIntelligenceDataRef()->getCursorTo()->getCursorPos())

{

if (getRules()->whatFractionSurrounds(getCurrCursor()->getCursorPos()) == playerFraction)

//проверяем окружает ли моя команда клетку

{

intelCon.directLogAppend("\_BFL\_INTELLIGENCE\_CREATE\_CQUARE");

setAction(\_BFL\_INTELLIGENCE\_CREATE\_CQUARE);//задаём комманду создания клетки

intelCon.getIntelligenceDataRef()->swapCursor(); //меняем курсор

return;

}

menuCommandReset(); //если не моя команда, то сбрасываем ход

return;

}

//если два курсора не находятся на одном месте

std::vector<vecI2d> possibleTurns = intelCon.getIntelligenceDataRef()->getPossiblePlayerPos();

std::vector<vecI2d>::iterator iter = possibleTurns.begin();

while (iter != possibleTurns.end())

{

if ((\*iter) == intelCon.getIntelligenceDataRef()->getCursorTo()->getCursorPos())

{

std::cout << "ff";

setAction(\_BFL\_INTELLIGENCE\_MOVE\_CELL\_CUSTOM); //даём команду на перемещение

intelCon.getIntelligenceDataRef()->swapCursor(); //меняем курсоры

intelCon.directLogAppend("\_BFL\_INTELLIGENCE\_MOVE\_CELL\_CUSTOM");

intelCon.getIntelligenceDataRef()->clearPossiblePos(); //очищаем доступные ходы

}

++iter;

}

return;

}

menuCommandReset(); //если ни одно условие не совпало, то сбрасываем ход

setAction(\_BFL\_INTELLIGENCE\_RESET);

}

void PersonIntelligence::chooseAction()

{

switch (intelCon.getGSimDataRef()->getGameSimMenuMode())

{

case \_GAMESIM\_MODE\_MOVE\_CELL\_UP:

{

setAction(\_BFL\_INTELLIGENCE\_CURSOR\_MOVE\_ACTION);

intelCon.getIntelligenceDataRef()->getCurrCusor()->crsUP();

brk;

}

case \_GAMESIM\_MODE\_MOVE\_CELL\_DOWN:

{

setAction(\_BFL\_INTELLIGENCE\_CURSOR\_MOVE\_ACTION);

intelCon.getIntelligenceDataRef()->getCurrCusor()->crsDOWN();

brk;

}

case \_GAMESIM\_MODE\_MOVE\_CELL\_LEFT:

{

setAction(\_BFL\_INTELLIGENCE\_CURSOR\_MOVE\_ACTION);

intelCon.getIntelligenceDataRef()->getCurrCusor()->crsLEFT();

brk;

}

case \_GAMESIM\_MODE\_MOVE\_CELL\_RIGHT:

{

setAction(\_BFL\_INTELLIGENCE\_CURSOR\_MOVE\_ACTION);

intelCon.getIntelligenceDataRef()->getCurrCusor()->crsRIGHT();

brk;

}

case \_GAMESIM\_MODE\_CELL\_ENTER:

{

menuCommandEnter();

brk;

}

case \_GAMESIM\_MODE\_CELL\_RESET:

{

menuCommandReset();

setAction(\_BFL\_INTELLIGENCE\_RESET);

brk;

}

default:

break;

}

}

#undef brk

**PersonIntelligence.h**

#ifndef \_PERSON\_INTELLIGENCE\_H\_

#define \_PERSON\_INTELLIGENCE\_H\_

#include "Intelligence.h"

#include "BattleForLifeGameComponents.h"

/\*

Класс интеллекта игрока-человека

Здесь находятся все необходимые для игры обработки

-комманд с клавиатуры

\*/

class PersonIntelligence : public Intelligence

{

private:

public:

PersonIntelligence(OEDataBase\* p\_dataBase, DBIntelligenceData\* dataRef, \_FRACTION\_INT\_ fraction, BattleForLifeRules\* rulesRef)

: Intelligence(p\_dataBase, dataRef, fraction, rulesRef)

{

}

~PersonIntelligence() {}

void intelligenceInit();

void chooseAction();

void menuCommandEnter();

void menuCommandMove();

void menuCommandReset();

void menuComEnterFromCursorAction();

};

#endif // !\_PERSON\_INTELLIGENCE\_H\_

**Player.cpp**

#include "Player.h"

#include "GameSimStructs.h"

#include "ArtificialIntelligence.h"

#define brk break

// Функция попытки сделать ход у игрока. Здесь выбираются варианты событий

bool Player::makeTurn()

{

//Выбираем действие

getMind()->chooseAction();

bool isSuccess = false;;

int action = getMind()->giveAction(); //Выдаём действие

switch (action) //выбираем из списка доступных действий

{

case \_BFL\_INTELLIGENCE\_MOVE\_CELL\_CUSTOM:

{

//Движение клетки из позиции первого курсора во второй. Временно клетки можно перемещать в любое место

isSuccess = rules->moveCell(

getFraction(),

getMind()->getCursorFrom()->getCursorPos(), getMind()->getCursorTo()->getCursorPos()

);

getMind()->getCursorFrom()->getCursorPtr()->setVectorI2D(

getMind()->getCursorTo()->getCursorPos()

);

//Обработка успешности перемещения клетки

if (true == isSuccess)

{

return true;

}

else

{

return false;

}

brk;

}

//Команда по созданию своей клетки

case \_BFL\_INTELLIGENCE\_CREATE\_CQUARE:

{

isSuccess = rules->createSquareByPressintEnter();

//Обработка успешности создания

if (true == isSuccess)

{

return true;

}

else

{

return false;

}

brk;

}

//Команда сброса курсоров

case \_BFL\_INTELLIGENCE\_RESET:

{

return false;

brk;

}

case \_BFL\_INTELLIGENCE\_CURSOR\_MOVE\_ACTION:

{

brk;

}

case \_BFL\_INTELLIGENCE\_NO\_ACTION:

{

brk;

}

case \_BFL\_INTELLIGENCE\_END\_OF\_TURN:

{

return true;

}

default:

break;

}

return false;

}

#undef brk

//Инициализация начальных значений игрока

void Player::playerInit(int intelligenceMode)

{

switch (intelligenceMode)

{

case \_PLAYER\_INTELLIGENCE\_MODE\_PERSON\_MIND:

{

//Выбор игрока - человека

setMind(

new PersonIntelligence(rules->getGSimCon()->getDBReference(),

intelDataRef,

playerFraction,

getBFLRules())

);

getMind()->intelligenceInit();

break;

}

case \_PLAYER\_INTELLIGENCE\_MODE\_COMPUTER\_MIND:

{

//выбор игрока - компьютера

setMind(

new ArtificialIntelligence(rules->getGSimCon()->getDBReference(),

intelDataRef,

playerFraction,

getBFLRules())

);

getMind()->intelligenceInit();

break;

}

default:

throw \_PLAYER\_INIT\_ERR\_INTELLIGENCE\_MODE\_UNDEF;

break;

}

}

**Player.h**

#ifndef \_PLAYER\_H\_

#define \_PLAYER\_H\_

#include "BattleForLifeGameComponents.h"

#include "PersonIntelligence.h"

#include "ArtificialIntelligence.h"

#define \_PLAYER\_INIT\_ERR\_INTELLIGENCE\_MODE\_UNDEF "PLAYER\_INIT\_ERR\_INTELLIGENCE\_MODE\_UNDEF"

#define \_PLAYER\_INTELLIGENCE\_MODE\_PERSON\_MIND 1

#define \_PLAYER\_INTELLIGENCE\_MODE\_COMPUTER\_MIND 2

/\*

Класс игрока

Здесь содержатся общие для игроков действия

А так же здесь содержится "тип интеллекта игрока"

\*/

class Player

{

private:

//PersonIntelligence mind;

Intelligence\* mind = NULL; //Интеллект грока. Или Человек, или компьютер

\_FRACTION\_INT\_ playerFraction; //команда игрока

BattleForLifeRules \*rules; //ссылка на правила игры

DBIntelligenceData\* intelDataRef; //ссылка на данные интеллекта

public:

void setPlayerFraction(\_FRACTION\_INT\_ newFraction)

{

playerFraction = newFraction;

}

Intelligence\* getMind()

{

return mind;

}

void setMind(Intelligence\* newMindPtr)

{

if (mind != NULL)

{

delete mind;

}

mind = newMindPtr;

}

\_FRACTION\_INT\_ getFraction()

{

return playerFraction;

}

Player(BattleForLifeRules\* p\_rules, DBIntelligenceData\* dataRef , \_FRACTION\_INT\_ fraction) :

//mind(p\_rules->getGSimCon()->getDBReference(), dataRef , playerFraction),

intelDataRef(dataRef),

rules(p\_rules), playerFraction(fraction)

{

std::cout << "Player constructor\n";

}

BattleForLifeRules\* getBFLRules()

{

return rules;

}

virtual ~Player()

{

delete mind;

}

//Каждый игрок обязан выбрать тип интеллекта

virtual void playerInit(int intelligenceMode);

//Каждый игрок обязан сделать ход

virtual bool makeTurn();

};

#endif // !\_PLAYER\_H\_

**Vector2DTemplate.h**

#ifndef VECTOR2DTEMPLATE\_H\_

#define VECTOR2DTEMPLATE\_H\_

#define \_DYNAMIC\_2D\_ARRAY\_TEMPLATE\_TYPE "\_DYNAMIC\_2D\_ARRAY\_TEMPLATE\_TYPE"

#include <vector>

#include <string>

template<class T>

class DynamicIntArray2DTemplated

{

private:

int sizeHorizonal{ 0 };

int sizeVertical{ 0 };

protected:

std::vector<std::vector<T>> dynamicVectorArray;

std::string type;

public:

DynamicIntArray2DTemplated()

{

setType(\_DYNAMIC\_2D\_ARRAY\_TEMPLATE\_TYPE);

DynamicIntArray2DTemplated::sizeHorizonal = 0;

DynamicIntArray2DTemplated::sizeVertical = 0;

}

void setType(std::string newType) { type = newType; }

void setHorizontal(int newHorizontal)

{

sizeHorizonal = newHorizontal;

}

void setVertical(int newVertical)

{

sizeVertical = newVertical;

}

int getSizeHorizonal() { return sizeHorizonal; }

int getSizeVertical() { return sizeVertical; }

long int getSurfaceArea() { return (sizeHorizonal + 1) \* (sizeVertical + 1); }

std::string getType() { return type; };

bool isCoordInLimits(int x, int y)

{

if (x < 0 || x >= getSizeHorizonal())

{

return false;

}

if (y < 0 || y >= getSizeVertical())

{

return false;

}

return true;

}

std::vector<std::vector<T>>\* getArray() { return &dynamicVectorArray; }

bool setElem(T newElem, int X, int Y)

{

if (getSizeHorizonal() < X || getSizeVertical() < Y || X < 0 || Y < 0)

{

return false;

}

dynamicVectorArray[Y][X] = newElem;

return true;

}

T\* getElem(int X, int Y)

{

T\* elem;

elem = &dynamicVectorArray.at(Y).at(X);

return elem;

}

std::vector<std::vector<T>>\* createNewArray(int newHorizontal, int newVertical)

{

if (dynamicVectorArray.size() != NULL)

{

deleteArr();

}

T tempElem;

for (int y = 0; y < newVertical; y++)

{

std::vector<T> tempVec;

for (int x = 0; x < newHorizontal; x++)

{

tempVec.push\_back(tempElem);

}

dynamicVectorArray.push\_back(tempVec);

}

DynamicIntArray2DTemplated::setHorizontal(newHorizontal);

DynamicIntArray2DTemplated::setVertical(newVertical);

return &dynamicVectorArray;

}

void fillArrayBy(T elem)

{

if (NULL == getArray())

return;

for (int y = 0; y < sizeVertical; y++)

{

for (int x = 0; x < sizeHorizonal; x++)

{

dynamicVectorArray[y][x] = elem;

}

}

}

void fillArrayBy()

{

T newElem = T();

fillArrayBy(newElem);

}

void deleteArr()

{

int tempX = sizeHorizonal;

int tempY = sizeVertical;

if (NULL == getArray())

return;

for (int y = 0; y < sizeVertical; y++)

{

dynamicVectorArray.at(y).clear();

}

dynamicVectorArray.clear();

DynamicIntArray2DTemplated::setHorizontal(0);

DynamicIntArray2DTemplated::setVertical(0);

}

void recreateArray()

{

int tempX = sizeHorizonal;

int tempY = sizeVertical;

if (NULL == getArray())

return;

for (int y = 0; y < sizeVertical; y++)

{

dynamicVectorArray.at(y).clear();

}

createNewArray(tempX, tempY);

fillArrayBy();

}

bool deleteElem(int X, int Y)

{

if (NULL == getElem(X, Y))

{

return false;

}

else

{

delete dynamicVectorArray[Y][X];

}

dynamicVectorArray[Y][X] = NULL;

return true;

}

};

#endif // !VECTOR2DTEMPLATE\_H\_

**IpsGame.cpp**

#include <iostream>

#include "GameCore.h"

#include <string>

#include <cstdio>

#include <cstdlib>

#include "Vector2DTemplate.h"

#include "GameBaseComponents.h"

int main()

{

//Непосредственно сам экземпляр игры

GameCore GC\_ex;

GCStartingArgs arg; //это dummy аргументы, которые здесь не играют роли

arg.testFlag = 1;

GC\_ex.GCstartMainFunc(arg); //запуск главного цикла игры

system("pause");

return 0;

}