МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Тихоокеанский государственный университет»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Проектирование и разработка игрового приложения “Морской бой”

Пояснительная записка к курсовой работе

КР. XXXXXXXXX.ТД

Выполнил студент Бояндин М.В.

Факультет, группа ФКФН, ПО(аб) – 81

Руководитель работы Федосеев А.А.

Виза: _____

(доработать, к защите и т.д.)

_____

Хабаровск – 2020г.

СОДЕРЖАНИЕ

1 Постановка задачи 3
2 Диаграмма классов 4
3 Описание классов 5
3.1 Класс cell 5
3.2 Класс emptyCell 5
3.3 Класс shipCell 6
3.4 Класс closedCell 6
3.5 Класс destroyedCell 6
3.6 Класс shotEmptyCell 7
3.7 Класс damagedCell 7
3.8 Класс field 7
3.9 Класс endingDialog 8
3.10 Класс player 8
3.11 Класс user 9
3.12 Класс bot 9
3.13 Класс shipCollection 9
3.14 Класс ship 10
3.15 Классы ship_1x1, ship_1x2, ship_1x3, ship_1x4 10
3.16 Класс shipManager 11
4 Диаграмма объектов 14
5 Диаграмма последовательностей и коммуникаций 15
6 Диаграммы состояний и переходов 16
7 Диаграмма модулей 18
Заключение 19
Список использованных источников 20
Приложение А 21
Приложение Б 26

1 ПОСТАНОВКА ЗАДАЧИ

Требуется спроектировать и разработать игровое приложение “Морской бой”, которое будет предоставлять пользователю возможность сыграть в одноименную игру против компьютера. Во время игры пользователю должно быть доступно общее число его кораблей и кораблей противника, клетки должны быть подсвечены так, чтобы игровая ситуация была интуитивна понятна. При завершении раунда пользователю следует предлагать возможность начать новый, либо выйти из игры. При разработке был использован фреймворк Qt 5.

2 ДИАГРАММА КЛАССОВ

Диаграмма классов в упрощенном виде приведена на рисунке 2.1. Полная диаграмма классов приведена в приложении Б.

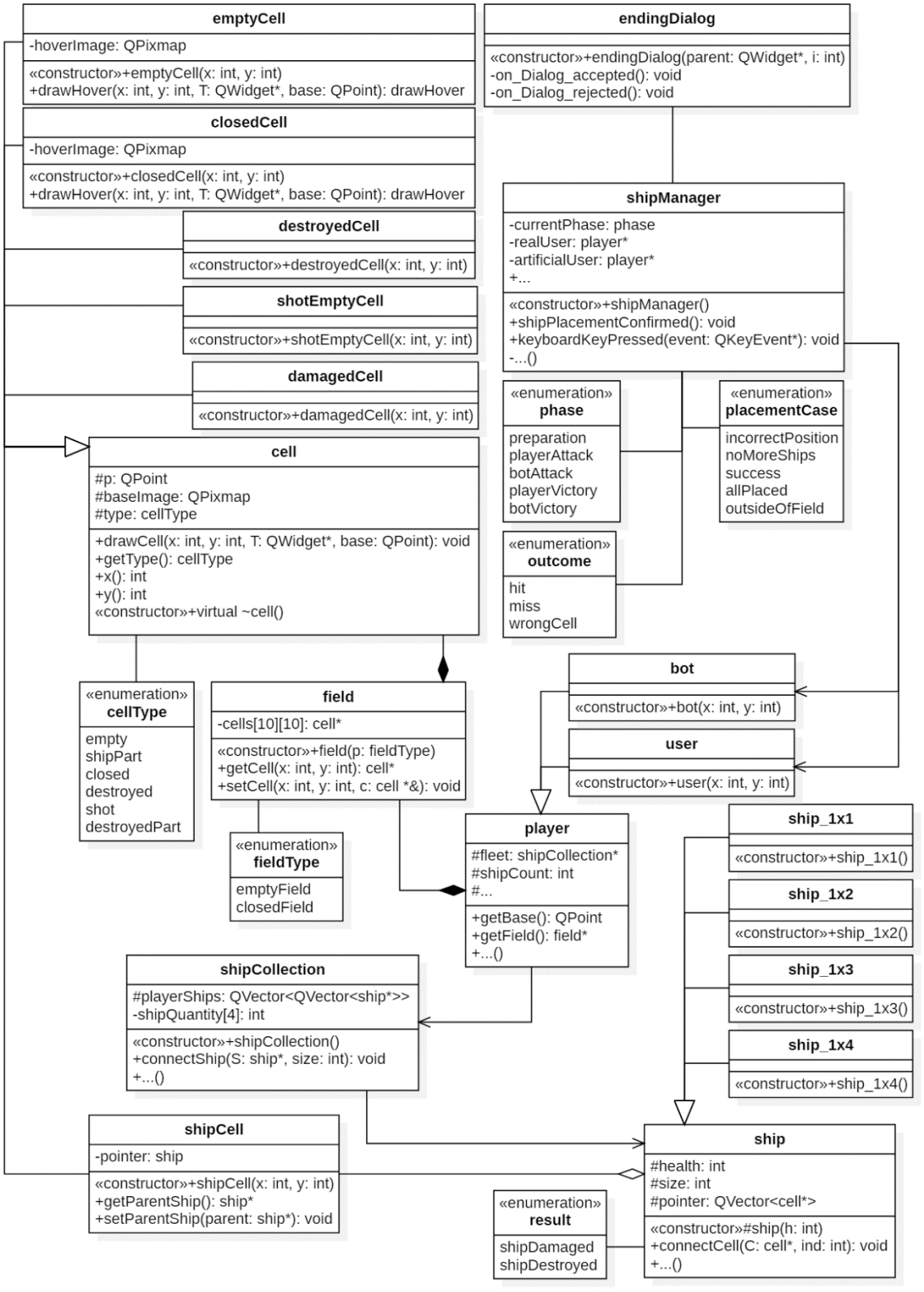

Рисунок 2.1 – Диаграмма классов

3 ОПИСАНИЕ КЛАССОВ

3.1 Класс cell

Класс cell описывает пустую клетку игрового поля.

Защищенные поля класса:

– QPoint p – содержит положение клетки;

– QPixmap baseImage – содержит изображение клетки;

– cellType type – содержит тип клетки.

Открытые методы класса:

– void drawCell(int x,int y, QWidget* T, QPoint base) – отображает клетку на поле;

– cellType getType() – позволяет получить тип клетки;

– int x() – возвращает координату *x* клетки;

– int y() – возвращает координату *y* клетки.

3.2 Класс emptyCell

Класс emptyCell является наследником класса cell. Описывает пустую клетку игрового поля.

Закрытые поля класса:

– QPixmap hoverImage – содержит изображение подсвеченной клетки.

Открытые методы класса:

– emptyCell(int x, int y) – создает пустую клетку с позицией (x, y);

– void drawHover(int x,int y, QWidget* T, QPoint base) – отображает «подсвеченное» изображение.

3.3 Класс shipCell

Класс shipCell является наследником класса cell. Описывает клетку-часть корабля.

Закрытые поля класса:

– ship* pointer – указатель на родительский корабль.

Открытые методы класса:

– shipCell(int x, int y) – создает клетку-часть корабля с позицией (x, y);

– ship* getParentShip() – возвращает указатель на корабль;

– void setParentShip(ship * parent) – устанавливает родительский корабль.

3.4 Класс closedCell

Класс closedCell является наследником класса cell. Описывает закрытую клетку игрового поля.

Закрытые поля класса:

– QPixmap hoverImage – содержит изображение подсвеченной клетки.

Открытые методы класса:

– closedCell(int x, int y) – создает закрытую клетку с позицией (x, y);

– void drawHover(int x,int y, QWidget* T, QPoint base) – отображает «подсвеченное» изображение.

3.5 Класс destroyedCell

Класс destroyedCell является наследником класса cell. Описывает клетку-часть уничтоженного корабля.

Открытые методы класса:

– destroyedCell(int x, int y) – создает уничтоженную клетку с позицией (x, y).

3.6 Класс shotEmptyCell

Класс shotEmptyCell является наследником класса cell. Описывает клетку, в которую выстрелили и которая оказалась пустой.

Открытые методы класса:

– shotEmptyCell(int x, int y) – создает клетку, в которую выстрелили и которая оказалась пустой, с позицией (x, y).

3.7 Класс damagedCell

Класс damagedCell является наследником класса cell. Описывает клетку-часть поврежденного корабля.

Открытые методы класса:

– damagedCell(int x, int y) – создает поврежденную клетку с позицией (x, y).

3.8 Класс field

Класс field описывает игровое поле.

Закрытые поля класса:

– cell* cells[10][10] – массив указателей на клетки данного поля.

Открытые методы класса:

– field(fieldType p) – создает поле соответствующего типа;

– cell* getCell(int x, int y) – возвращает указатель на клетку с позицией (x, y);

– void setCell(int x, int y, cell *&c) – устанавливает клетку на поле на позицию (x, y) и удаляет старую клетку.

3.9 Класс endingDialog

Класс endingDialog описывает диалог после завершения раунда.

Открытые методы класса:

– void on_Dialog_accepted() – перезапускает приложение и начинает новую игру;

– void on_Dialog_rejected() – закрывает приложение.

3.10 Класс player

Класс player описывает игрока.

Закрытые поля класса:

– shipCollection* fleet – содержит указатель на флот игрока;

– int shipCount – содержит общее число кораблей игрока;

– QPoint basePoint – содержит точку для отрисовки поля игрока;

– field* F – содержит указатель на поле игрока;

– field* fieldForDisplay – содержит указатель на поле игрока, которое видит противник.

Открытые методы класса:

– QPoint getBase() – возвращает точку для отрисовки поля игрока;

– field* getField() – возвращает указатель на поле игрока;

– field* getFieldForDisplay() – возвращает указатель на поле игрока, которое видит противник;

– void setFleet(shipCollection* Fl) – устанавливает флот игрока;

– shipCollection* getFleet() – возвращает указатель на флот игрока;

– void incShipCount() – увеличивает общее количество кораблей;

– void decShipCount() – уменьшает общее количество кораблей;

– int getShipCount() – возвращает общее количество кораблей.

3.11 Класс user

Класс user является наследником класса player. Описывает пользователя-игрока.

Открытые методы класса:

– user(int x, int y) : player() – создает пользователя-игрока с заданной ему точкой для отрисовки поля.

3.12 Класс bot

Класс bot является наследником класса player. Описывает противника.

Открытые методы класса:

– bot(int x, int y) : player() – создает противника-компьютер с заданной ему точкой для отрисовки поля.

3.13 Класс shipCollection

Класс shipCollection описывает флот игрока.

Закрытые поля класса:

– QVector < QVector<ship*> > playerShips – вектор векторов, содержащих корабли одного размера;

– int shipQuantity[4] – массив с количеством кораблей по палубам.

Открытые методы класса:

– shipCollection() : playerShips(4) – создает экземпляр класса и устанавливает размер вектора равным 4;

– void connectShip(ship* S, int size) – добавляет корабль, на который указывает S, во флот;

– int getShipQuantity(int size) – возвращает количество кораблей данного размера во флоте;

– QVector<ship*>* getShipCollection(int size) – возвращает вектор кораблей данного размера;

– void incQuantity(int size) – увеличивает количество кораблей данного размера;

– void decQuantity(int size) – уменьшает количество кораблей данного размера.

3.14 Класс ship

Класс ship описывает корабль на игровом поле.

Защищенные поля класса:

– int health – хранит прочность корабля;

– int size – хранит размер корабля;

– QVector <cell*> pointer – хранит вектор указателей на части корабля.

Открытые методы класса:

– ship (int h) – создает корабль размера и прочности h;

– void connectCell(cell *C, int ind) – прикрепляет клетку к кораблю;

– int getHealth() – возвращает прочность корабля;

– cell* getShipPart(int ind) – возвращает часть корабля по индексу;

– int getSize() – возвращает размер корабля;

– QVector <cell*>* getAllShipParts() – возвращает все части корабля;

– result decHealth() – уменьшает прочность корабля.

3.15 Классы ship_1x1, ship_1x2, ship_1x3, ship_1x4

Данные классы являются наследниками класса ship. В своем конструкторе используют параметризованный конструктор родительского класса.

3.16 Класс shipManager

Класс shipManager описывает «управляющего» игрой – он отвечает за все процессы, связанные с взаимодействием пользователя с объектами игры.

Закрытые поля класса:

– phase currentPhase – содержит текущую фазу игры;

– player* realUser – содержит указатель на игрока-пользователя;

– player* artificialUser – содержит указатель на игрока-компьютер;

– QPoint* mPosPrev – содержит указатель на предыдущее положение курсора мыши, приведенное к полю;

– QPoint mPos – содержит указатель на текущее положение курсора мыши;

– QPoint sPos[4] – содержит координаты текущего корабля для отрисовки;

– int shipSize – содержит размер текущего корабля;

– int rotation – содержит положение текущего корабля;

– bool cursorOutOfBounds - указывает, находится ли курсор в пределах игрового поля.

Открытые методы класса:

– shipManager() – создает объект класса и для него создает объекты игрока и противника с соответствующими им точками для отрисовки, а также устанавливает корабли для противника void shipPlacementConfirmed() – обрабатывает ситуацию, когда пользователь подтвердил расстановку своих кораблей: устанавливает фазу атаки пользователя и фиксирует размер и положение корабля;

– void keyboardKeyPressed(QKeyEvent *event) – обрабатывает нажатие клавиши пользователем, если нажата клавиша “R”, то осуществляет поворот корабля;

– void wheelScrolled(QWheelEvent *event) – обрабатывает прокрутку колеса мыши: если колесо прокручено от пользователя, то увеличивает размер устанавливаемого корабля на 1, иначе уменьшает размер устанавливаемого корабля на 1;

– player* getUser() – возвращает указатель на игрока-пользователя;

– player* getBot() – возвращает указатель на игрока-противника;

– void userAttack(QWidget *T) – обрабатывает атаку пользователя: при успешном попадании возвращает управление пользователю, иначе устанавливает фазу атаки противника и вызывает функцию AIAttack. Если кораблей не осталось, то устанавливает фазу победы игрока и вызывает финальный диалог;

– void setPhase(phase P) – устанавливает фазу игры;

– void deleteShip(player *owner) – удаляет корабль owner, на который указывает курсор мыши;

– placementCase setShip(player *owner) – устанавливает корабль на игровое поле owner. Если все корабли данного размера установлены, то возвращает noMoreShips. Если не все клетки подходят для установки корабля, то возвращает incorrectPosition. Если установлены все корабли, то возвращает allPlaced, если не все и установка была успешной – success. В случае, когда клик мыши был произведен за пределами поля – возвращает outsideOfField.

Закрытые методы класса:

– bool checkForRemainingShips(player* opponent) – проверяет, остались ли корабли у игрока, на которого указывает opponent. Если остались, то возвращает true, иначе – false;

– void finalDialog() – создает диалог и вызывает его поверх всех окон.

– void delay(int ms) – обеспечивает задержку выполнения программного кода на ms миллисекунд;

– void AIAttack(QWidget *T) – обрабатывает атаку компьютера: при успешном попадании продолжает атаку, иначе устанавливает фазу атаки

игрока и передает управление в функцию userAttack. Если кораблей не осталось, то вызывает фазу победы противника и вызывает финальный диалог;

– outcome fireAtShip(player *recipient) – обрабатывает атаку на корабль игрока recipient. При попадании и повреждении корабля заменяет соответствующую клетку корабля, уменьшает его прочность и возвращает hit. При попадании и уничтожении корабля заменяет все его клетки на destroyedCell, уменьшает общее число кораблей recipient, удаляет корабль из флота и возвращает hit. При промахе заменяет соответствующую клетку поля на shotEmptyCell и возвращает miss;

– void xRAYbotField() – выводит консоль поле противника-компьютера;

– void autoSetShip() – устанавливает корабли противника на поле;

– bool isCellSuitable(QPoint C, player* owner) – проверяет, подходит ли клетка для установки, если подходит, то возвращает true, иначе false;

– void rotateShip(int rotation) – устанавливает поворот корабля;

– void updateHighlight() – обновляет подсветку корабля для отрисовки;

– void moveToCellLocation(QPoint basePos, QPoint mouse_pos) – перемещает подсветку на клетку, на которую указывает курсор мыши;

– phase getPhase() – возвращает текущую фазу игры;

– void drawBaseFields(QWidget *T) – рисует основные игровые поля;

– void drawBattle(QWidget *T) – рисует подсветку на полях в соответствии с текущей фазой игры и параметрами корабля.

4 ДИАГРАММА ОБЪЕКТОВ

На рисунке 4.1 изображен сценарий установки четырехпалубного корабля на игровое поле. Все объекты корабля заменяются в игровом поле с помощью метода replaceCells. У пользователя увеличивается счетчик кораблей, в то время как в sc уменьшается количество кораблей данного размера. Объекты c[0][0..3] заносят в себя информацию о корабле s при помощи метода connectCell. Сам корабль и его клетки включаются в sc через метод connectShip класса shipCollection.

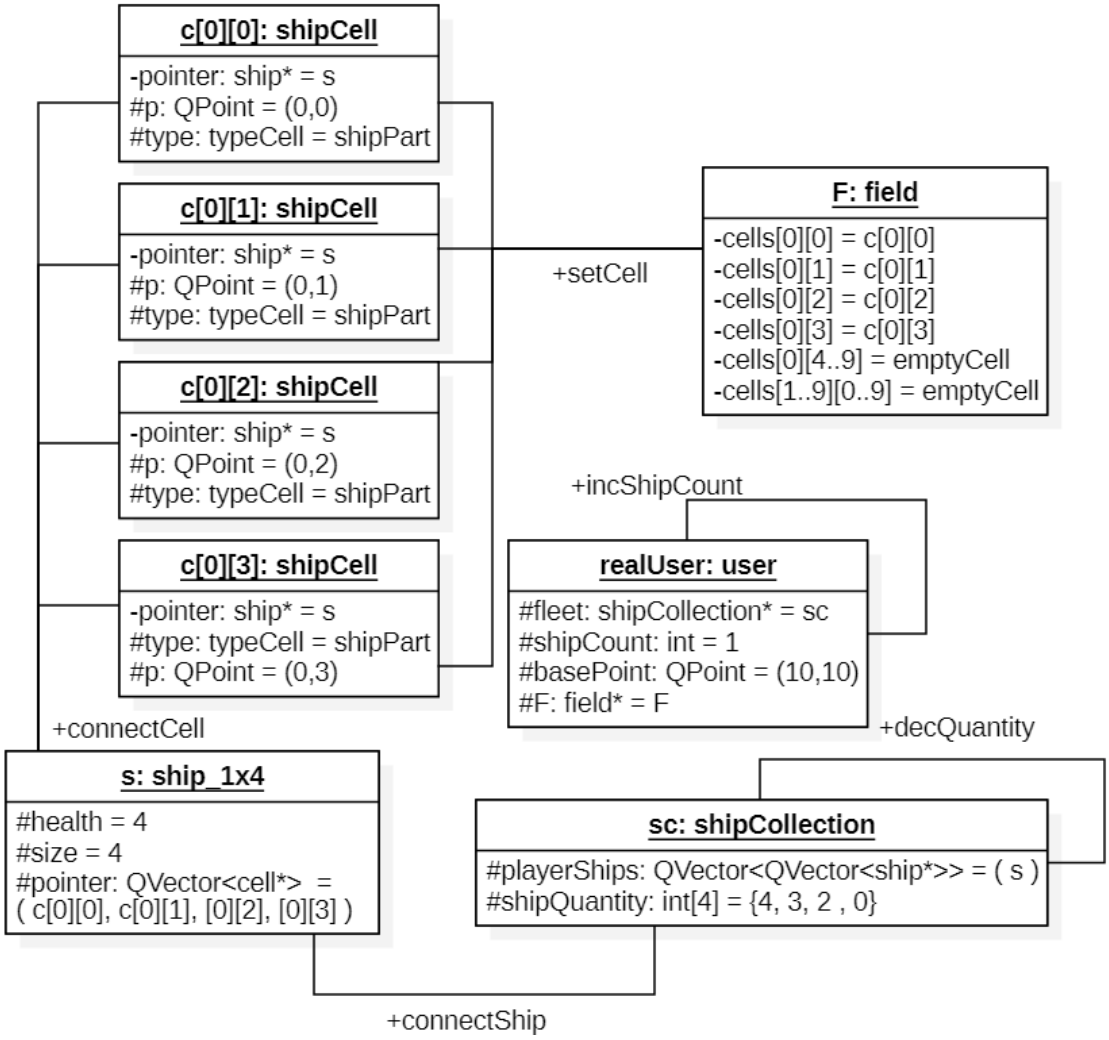

Рисунок 4.1 – диаграмма объектов

5 ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТЕЙ И КОММУНИКАЦИЙ

На рисунке 5.1 приведен сценарий установки пользователем однопалубного корабля. В полном разрешении диаграмма приведена в приложении Б. При нажатии мыши вызывается событие mousePressEvent, в котором проверяется текущая фаза игры, если это подготовка, то вызывается метод setShip класса shipManager. В этом методе сначала происходит проверка на доступность таких кораблей для установки. Далее создается экземпляр класса shipCell и экземпляр класса ship_1x1, к которому на следующем шаге прикрепляется temp. После этого клетка с кораблем устанавливается на поле владельца (сообщение setCell), на последнем шаге корабль включается в коллекцию кораблей игрока методом connectShip, после этого происходит возврат управления пользователю.

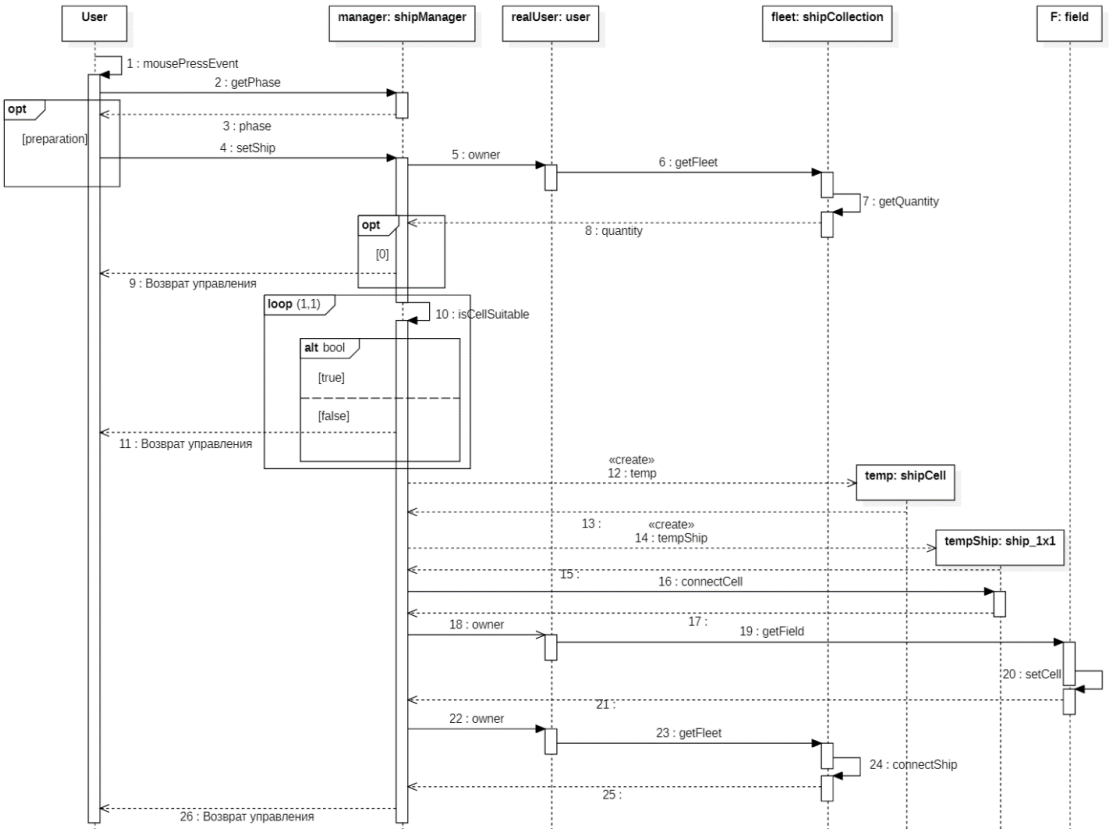

Рисунок 5.1 – диаграмма последовательностей и коммуникаций

6 ДИАГРАММЫ СОСТОЯНИЙ И ПЕРЕХОДОВ

На рисунке 6.1 представлена диаграмма состояний и переходов системы во время установки корабля. Изначально система находится в режиме ожидания, а переход в другое состояние «обработка нажатия», осуществляется с помощью клика мыши по игровому полю. После этого происходит проверка фазы игры, если фаза не соответствует «подготовке», то система возвращается в состояние ожидания. В случае, когда фаза игры удовлетворяет условию, система переходит в состояние «установка корабля». После происходит проверка, была ли установка успешной (данная схема упрощена). Если нет, то осуществляется переход в состояние ожидания, иначе система переходит в состояние обновления полей и после в ожидание действий пользователя.

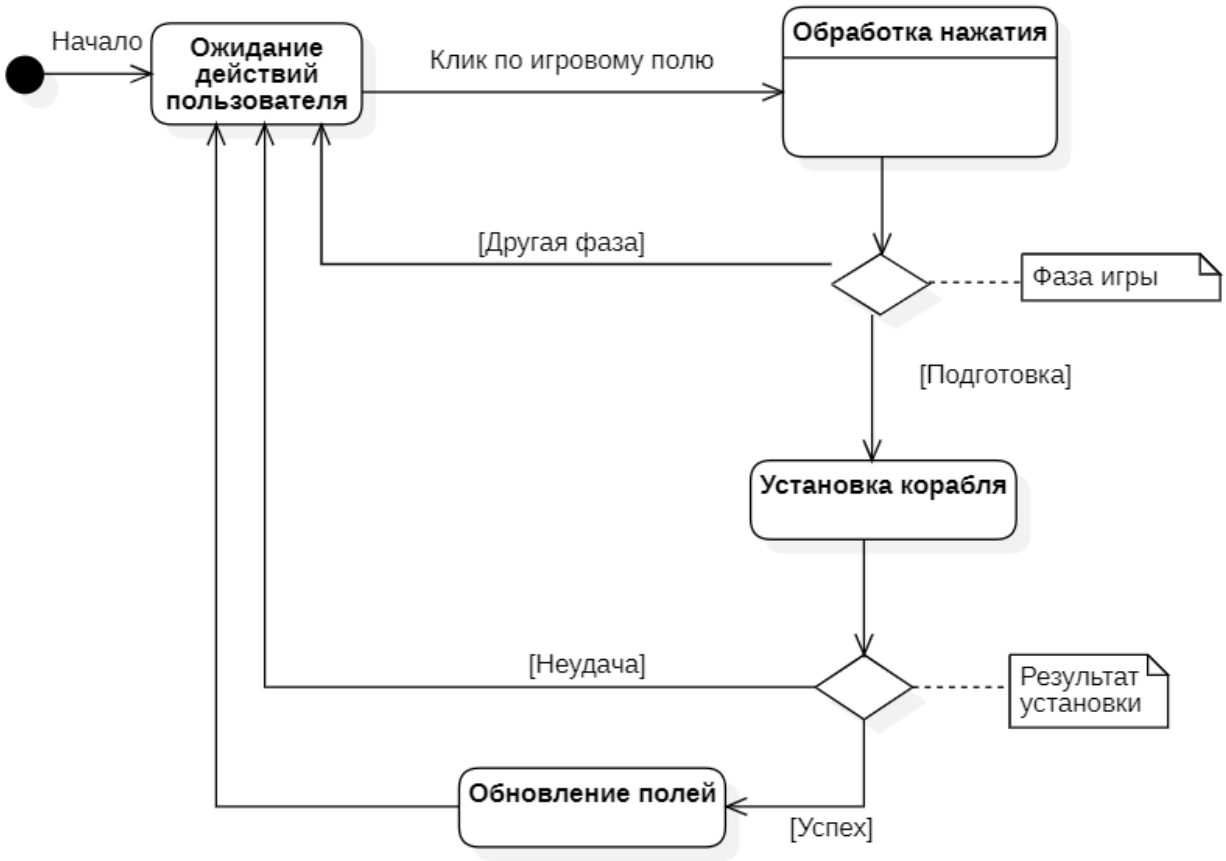

Рисунок 6.1 – Диаграмма состояний № 1

Диаграмма состояний, приведенная на рисунке 6.2, описывает основные состояния программного средства во время его использования. После запуска система принимает состояние расстановки кораблей, когда пользователь подтверждает расстановку, состояние сменяется на бой, который начинается с хода игрока.

В фазе игрока производится выстрел: если было попадание, то система сохраняет свое состояние, иначе она переходит в состояние «фаза противника». Аналогичная ситуация и для фазы противника. В каждом из этих двух состояний происходит тест на наличие кораблей у противника. Состояние «финал победа» достигается, если у противника нет кораблей. Состояние «финал поражение» достигается, если на фазе противника тест кораблей игрока выдал «нет кораблей». Оба состояния ведут к состоянию системы под названием «финальный диалог». Если пользователь выберет «выход», то приложение завершится, если «новая игра», то приложение будет перезапущено и система примет состояние «расстановка кораблей».

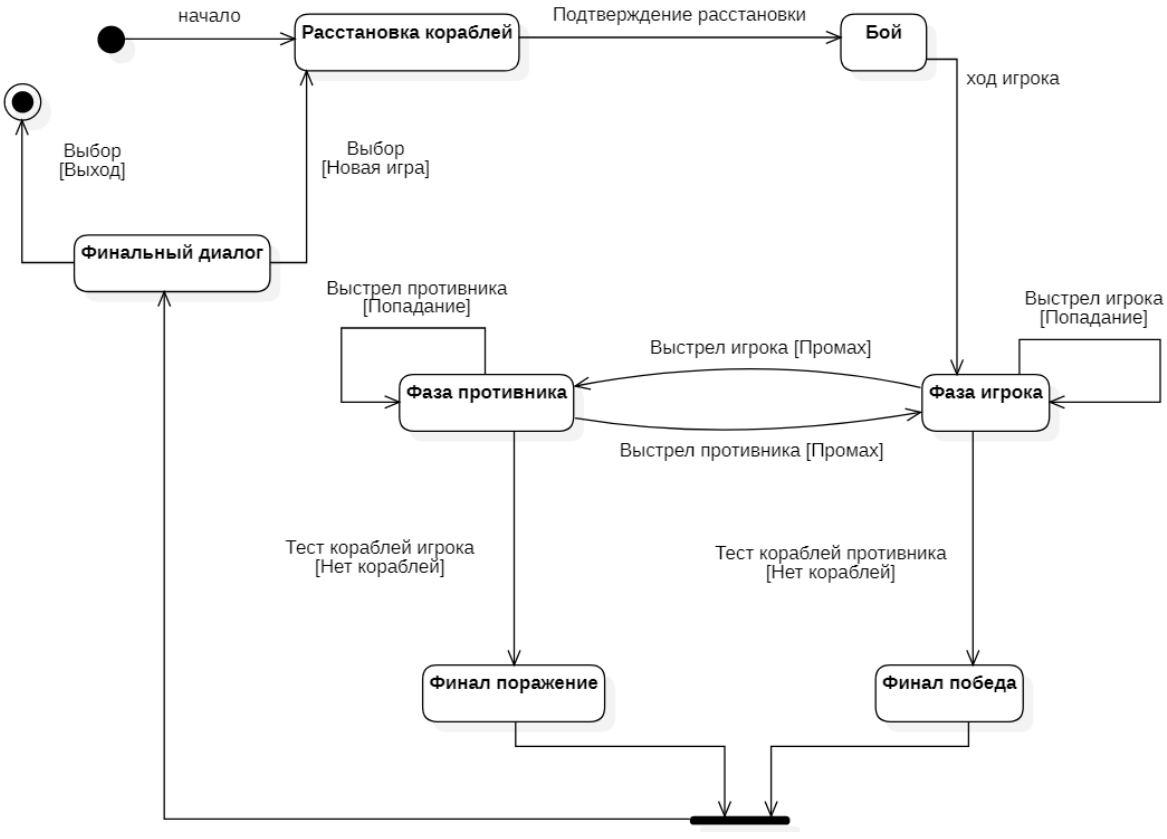

Рисунок 6.2 – Диаграмма состояний № 2

7 ДИАГРАММА МОДУЛЕЙ

На рисунке 7.1 представлена диаграмма модулей разработанного программного средства. В заголовочный файл fields.h подключены файлы ships.h и cells.h для отображения их в приложении и определения базовых классов для игры. Файл fields.h включен в shipCollection.h, так как в этом файле описан класс для флота игрока. shipCollection.h включается в player.h, т.к. флот принадлежит игроку. В shipManager.h, главный заголовочный файл приложения, включаются файлы player.h и interfaceDialog.h. shipManager.h включается в mainwindow.cpp.

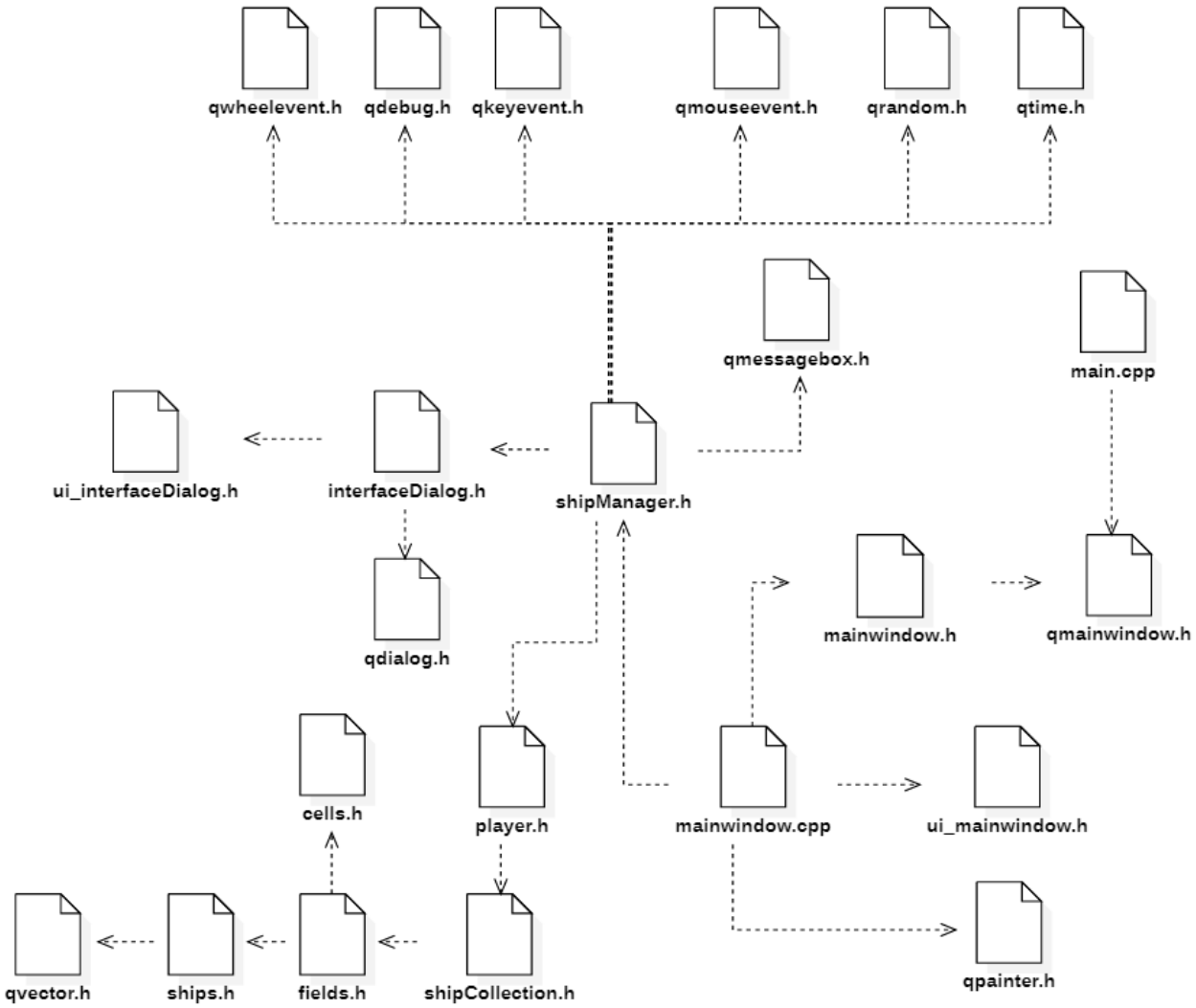

Рисунок 7.1 – диаграмма модулей

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы создано игровое приложение “Морской бой”, которое будет предоставлять пользователю возможность сыграть в одноименную игру против компьютера.

При разработке приложения усовершенствованы навыки разработки интерфейса для графического приложения и написания программного кода для обработки событий. Также закреплены принципы объектно-ориентированного программирования и навыки работы с фреймворком Qt 5.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя. 2-е изд.: Пер. с англ. Мухин Н. – М.: ДМК Пресс, 2006. – 496 с.: ил.

2. Лафоре Р. Объектно-ориентированное программирование в C++. Классика Computers Science. 4-е изд. — СПб.: Питер, 2018. — 928 с.: ил.

3. Qt Core 5.15.2 [Электронный ресурс]: официальный сайт фреймворка Qt. – Режим доступа: https://doc.qt.io/qt-5/qtcore-index.html (дата обращения 01.12.2020).

4. Шилдт Г. Самоучитель C++ : пер. с англ. - 3-е изд., перераб. и доп. - Санкт-Петербург : BHV-Сант-Петербург, 2005. - 688с. : ил.

ПРИЛОЖЕНИЕ А

(обязательное)

Скриншоты работы программы

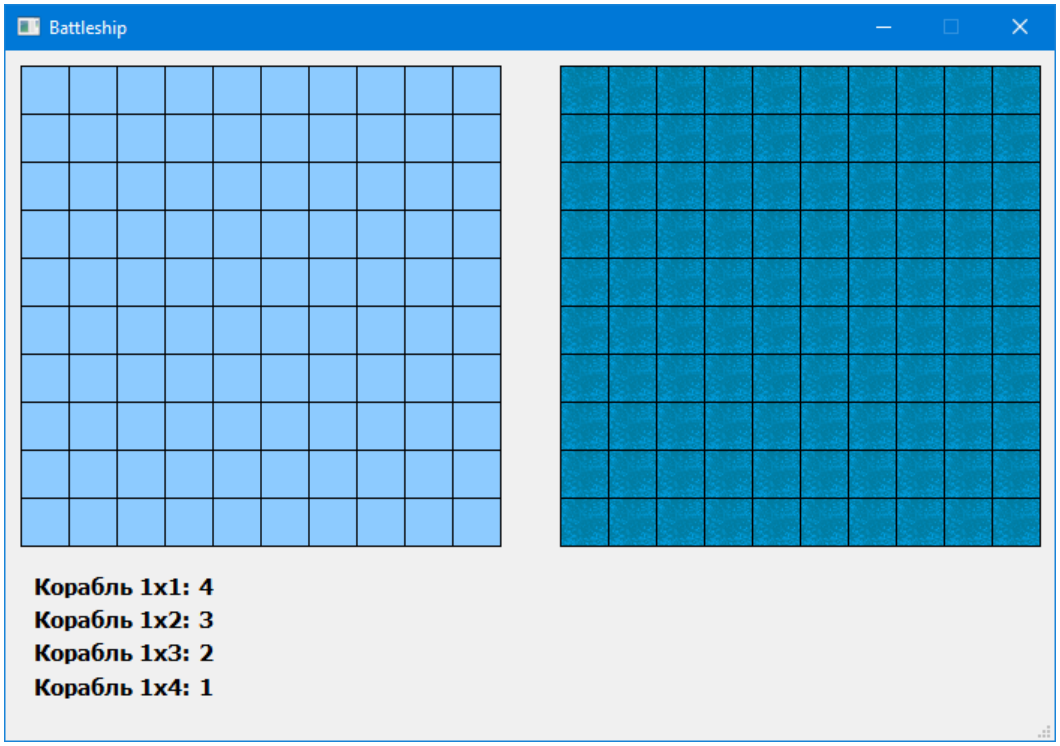

Рисунок А.1 – Начальное окно приложения

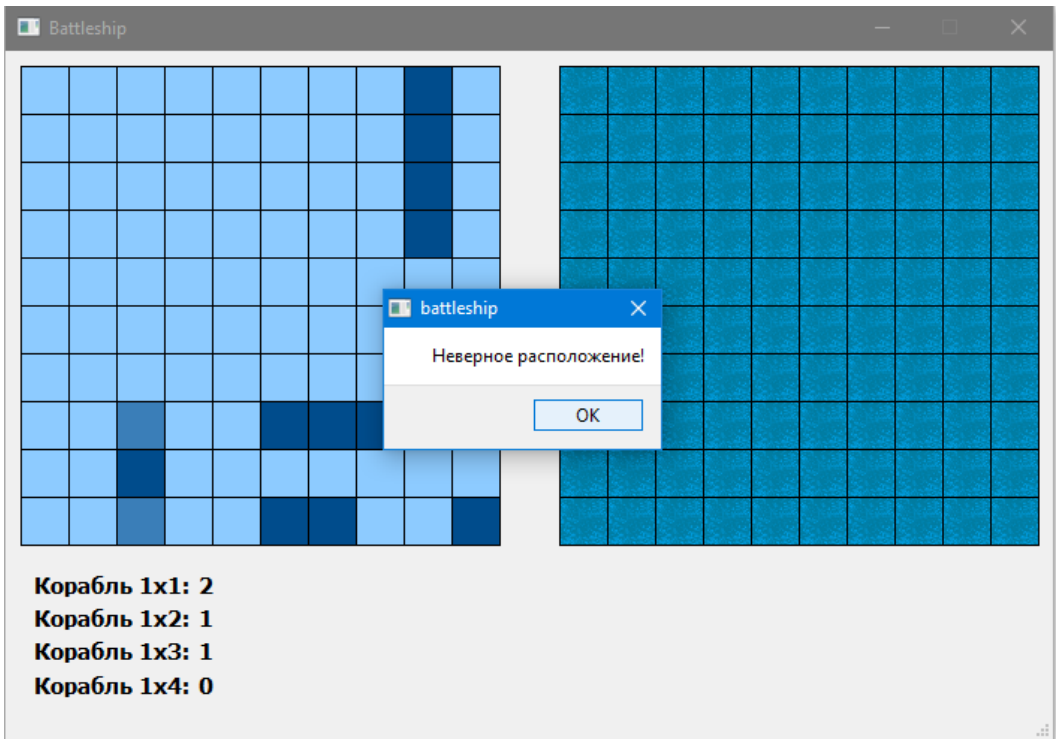

Рисунок А.2 – Неверное расположение корабля

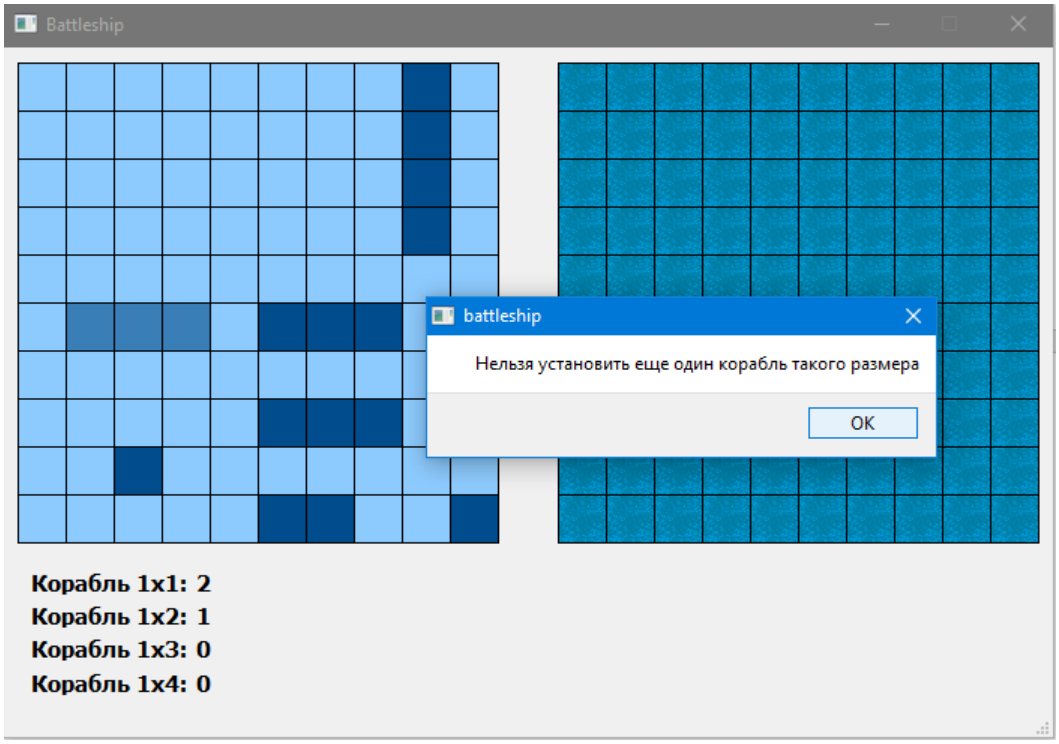

Рисунок А.3 – Попытка установить лишний корабль

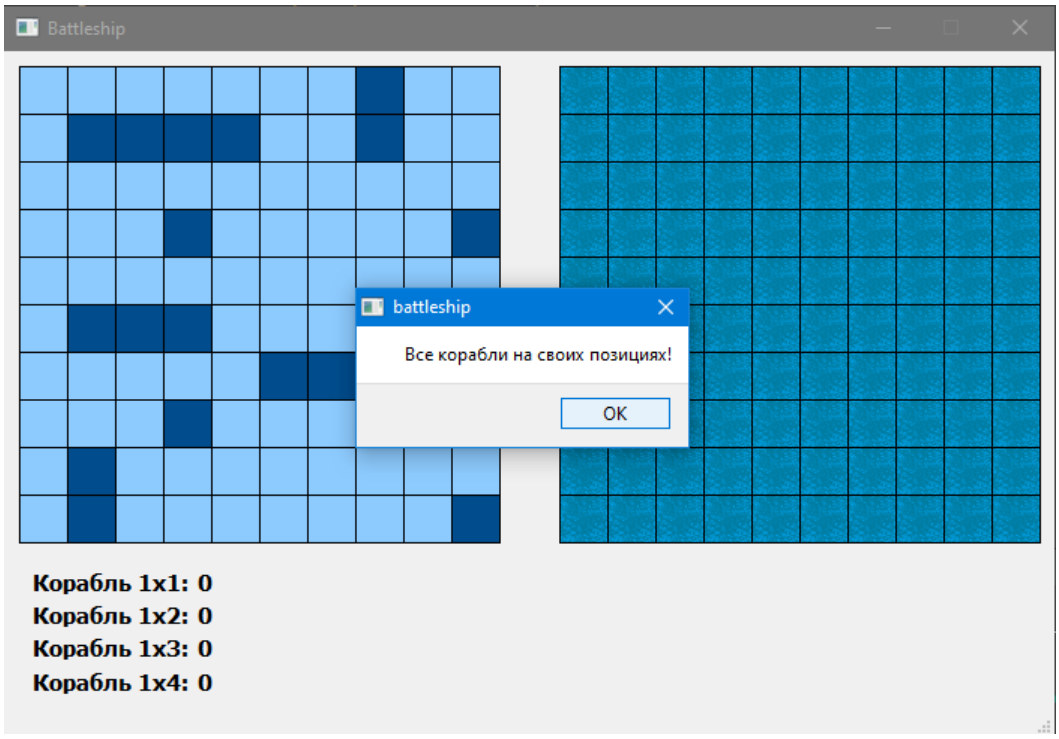

Рисунок А.4 – Все корабли на позициях

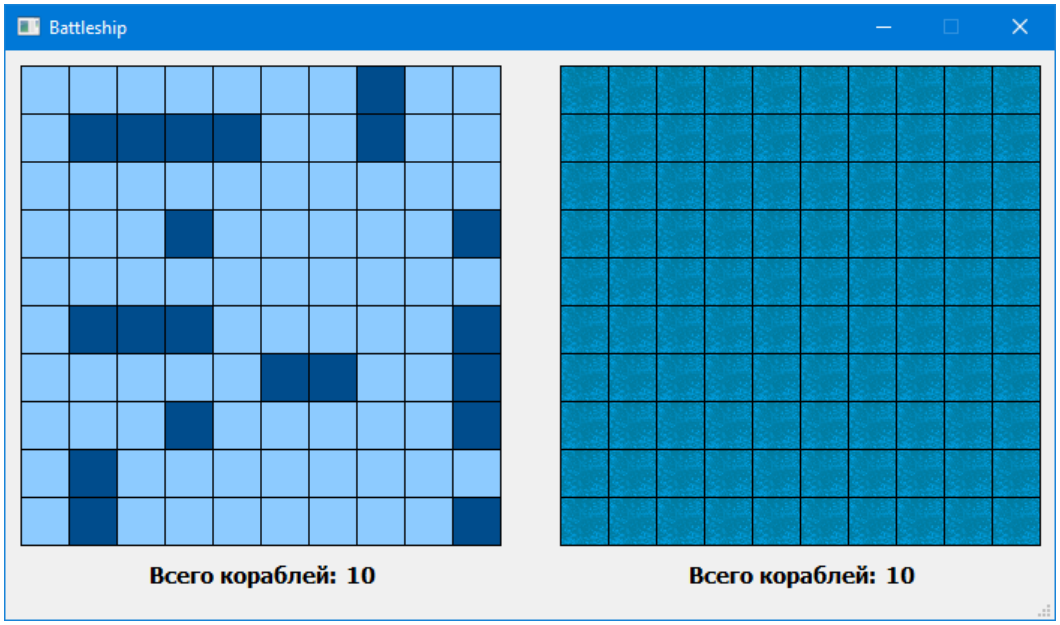

Рисунок А.5 – Игровое окно

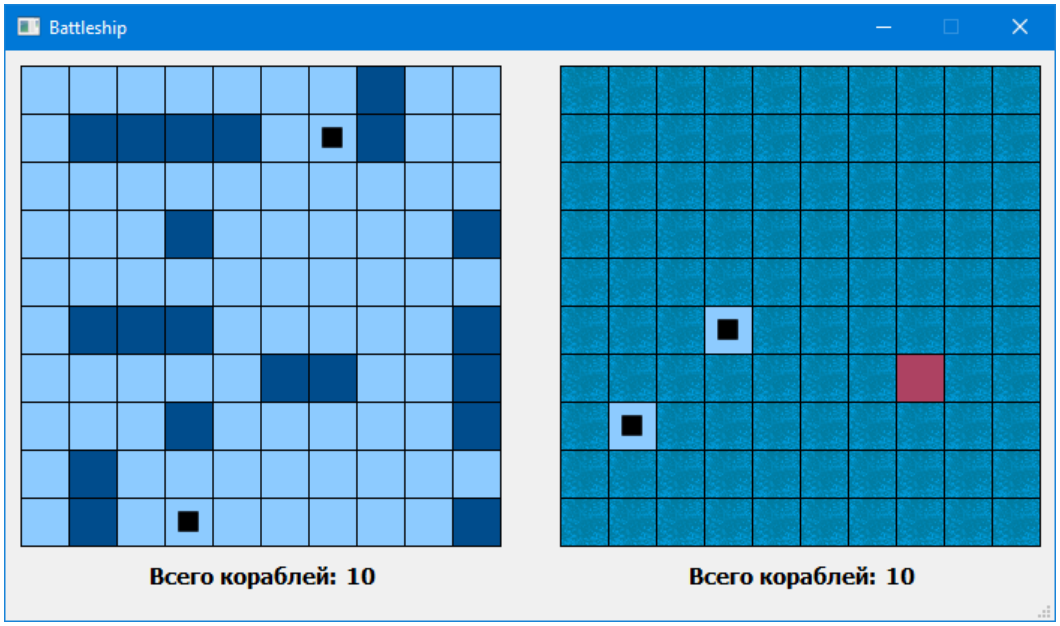

Рисунок А.6 – Промахи и маркер прицела

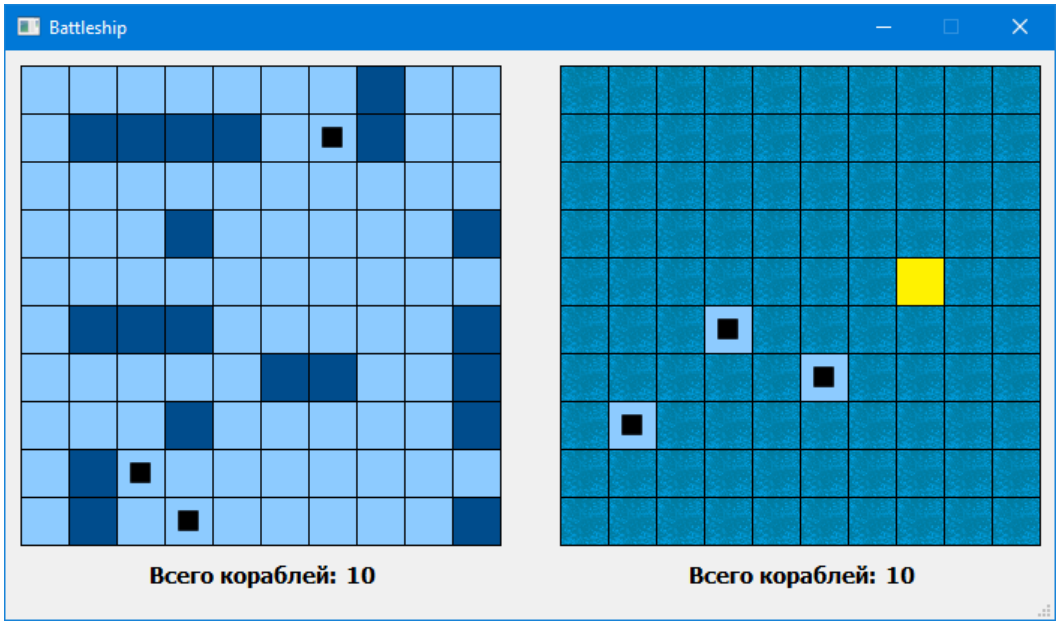

Рисунок А.7 – Попадание в корабль

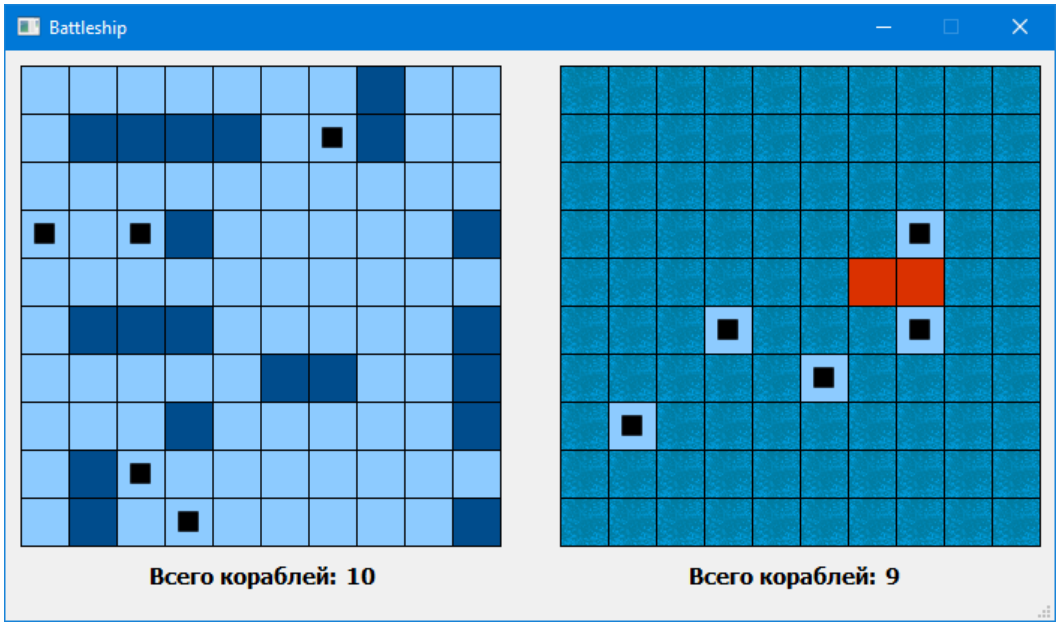

Рисунок А.8 – Уничтожение корабля

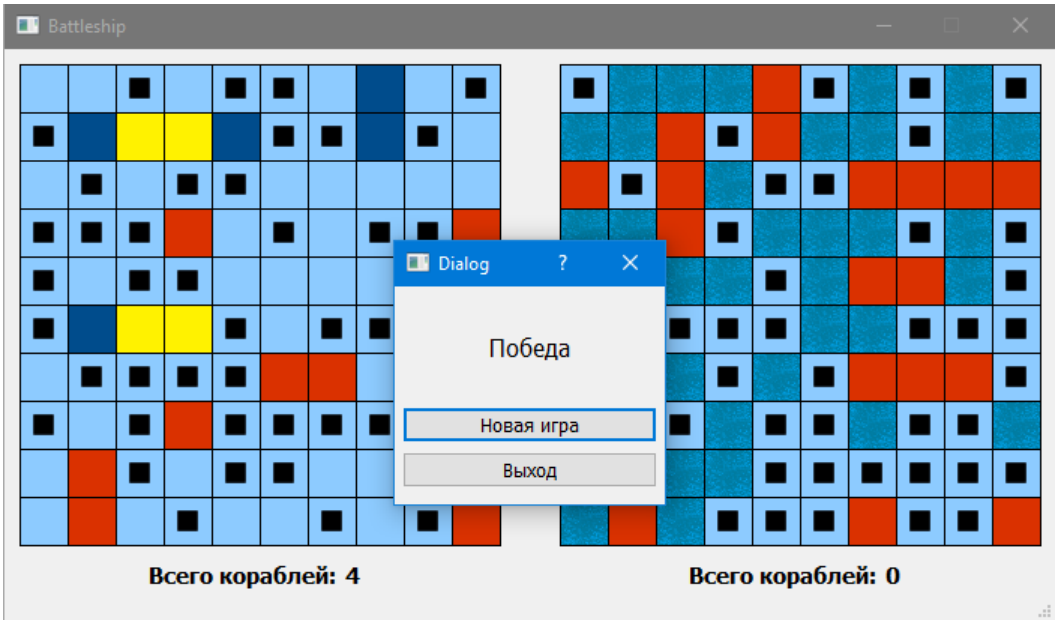

Рисунок А.10 – Победа игрока

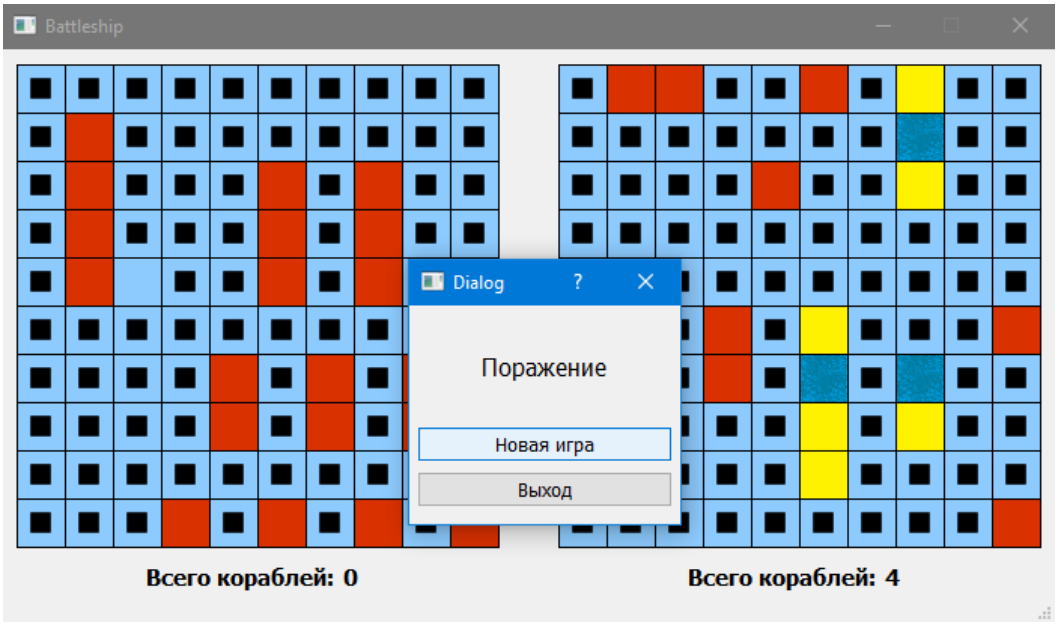

Рисунок А.11 – Поражение игрока

(обязательное)

Диаграммы

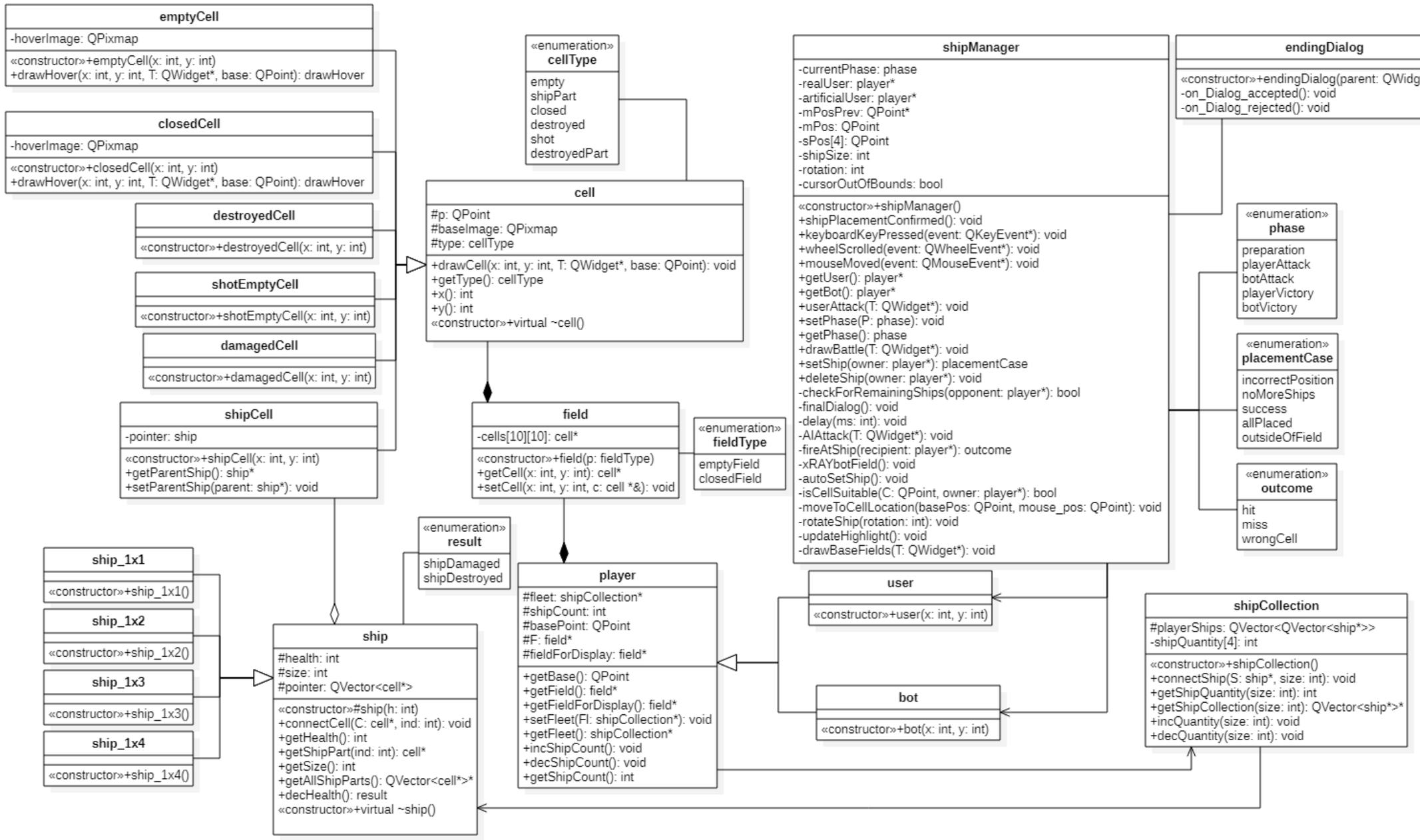

Рисунок Б.1 – Развернутая диаграмма классов

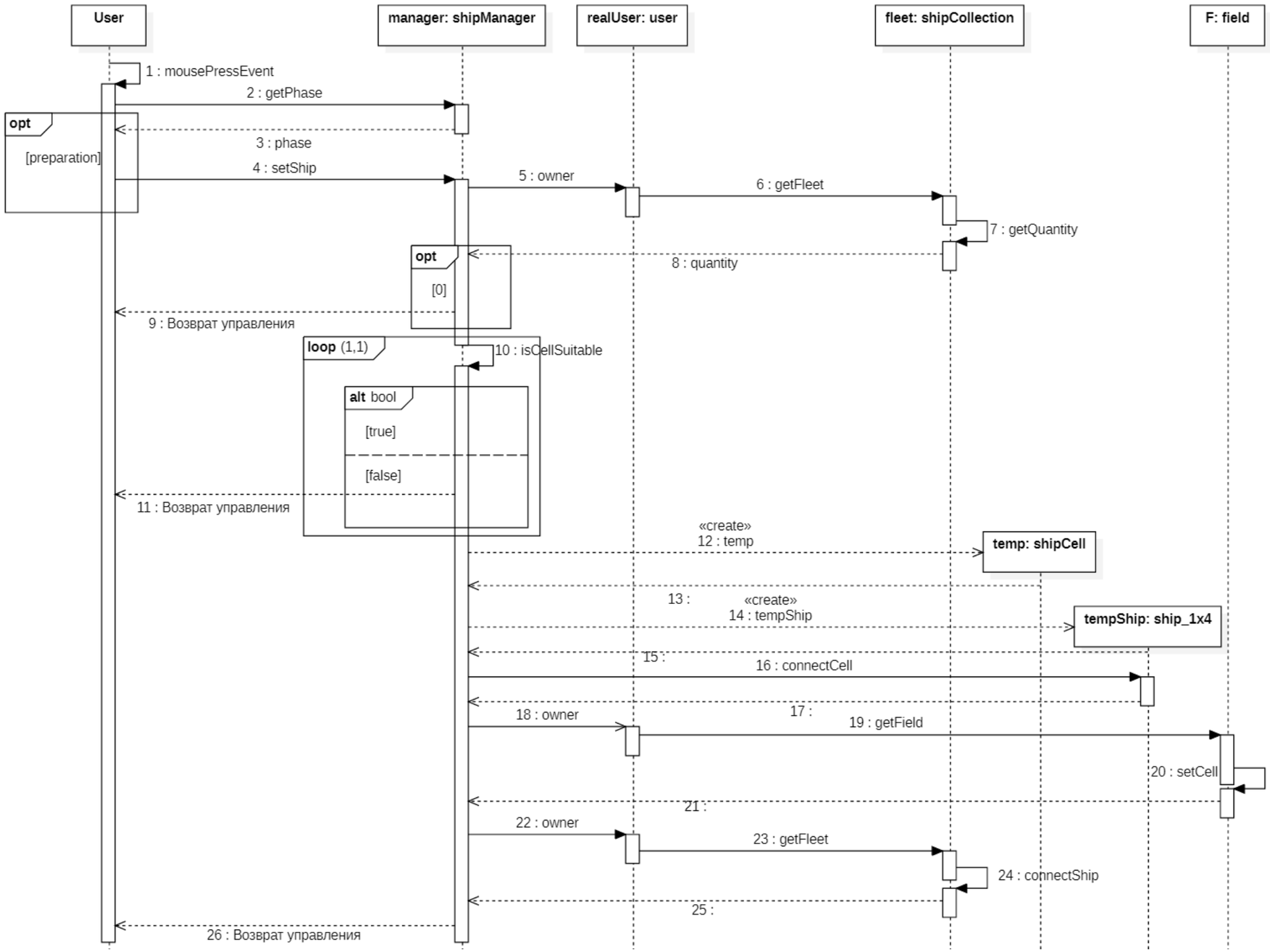

Рисунок Б.2 – Развернутая диаграмма последовательностей и коммуникаций