|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ | | | | |
| федеральное государственное бюджетное образовательное учреждение | | | | |
| высшего образования | | | | |
| «Балтийский государственный технический университет | | | | |
| «ВОЕНМЕХ» им. Д.Ф. Устинова» | | | | |
| (БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова») | | | | |
| БГТУ.СМК-Ф-4.2-К5-01 | | | | |
| Факультет | | | И |  | Информационные и управляющие системы |
|  |  |  | шифр |  | наименование |
| Кафедра | | | И5 |  | Информационные системы и программная инженерия |
|  |  |  | шифр |  | наименование |
| Дисциплина | |  | Программирование на языке высокого уровня | | |

КУРСОВАЯ РАБОТА

на тему

ИГРА “СОКОБАН”

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Выполнил студент группы | | | | | | |  | И597 |
|  |  | Апарин С.В. | | | | | | |
|  |  | Фамилия И.О. | | | | |  |  |
|  |  | **РУКОВОДИТЕЛЬ** | | | | | | |
| Спирин Д.О. | | | | | | |  |  |
| Фамилия И.О. | | | | |  |  | Подпись | |
| Оценка | | | | | | |  |  |
| « » | | |  |  |  |  |  | 2020 г. |

# СОДЕРЖАНИЕ

СОДЕРЖАНИЕ 2

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 3

ВВЕДЕНИЕ 4

1 Постановка задачи 5

2 Создание приложения 6

2 Создание основных компонентов 7

3 Организация программы 25

ЗАКЛЮЧЕНИЕ 27

ПРИЛОЖЕНИЕ 29

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем отчете применяют следующие сокращения и обозначения.

**SDL2** – кроссплатформенная библиотека разработки, предназначенная для обеспечения низкоуровневого доступа к аудио, клавиатуре, мыши, джойстику и графическому оборудованию через OpenGL и Direct3D

**STL** - набор согласованных обобщённых алгоритмов, контейнеров, средств доступа к их содержимому и различных вспомогательных функций в C++.

**UML** - язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

# ВВЕДЕНИЕ

Целью курсовой работы является создание собственной игры на основе объектно-ориентированного программирования, используя графическую библиотеку SDL версии 2.

Для своей курсовой работы я решил разработать игру «Сокобан» с

графическим интерфейсом, где игроку предлагается управлять персонажем с помощью клавиатуры.

Для окончания игры игроку необходимо расставить ящики по обозначенным местам лабиринта. Кладовщик одновременно может двигать только один ящик, толкая его вперёд.

**1 Постановка задачи**

Игра «Сокобан». Для классического варианта (ящики можно только толкать) в зависимости результатов объектно-ориентированного анализа и проектирования и пользовательского интерфейса – сложность от 2 до 3

Дополнительное правило – ящик можно не только толкать, но и тянуть – сложность 4 Дополнительное условие – на толкание или тягание ящиков расходуется энергия, которая восстанавливается за определенное время. Цель – не только поставить все ящики на места, но и сделать это за минимальное время.

Сложность 5 Энергия расходуется исходя из массы ящика, ящики при одинаковом размере могут иметь разную массу (написана на ящике). Скорость восстановления энергии зависит от скорости ее расхода (развиваемой мощности) и времени непрерывного расхода: чем с большей скоростью расходовалась энергия и/или чем больше было время расхода («работы без отдыха»), тем медленнее она восстанавливается (больше времени требуется на «отдых»). Сложность 5\*.

# 2 Создание приложения

Создание приложения можно разделить на несколько этапов:

1. На первом этапе необходимо ознакомиться с поставленной задачей и неформально описать решение задачи.
2. На втором этапе следует ознакомиться с инструментами разработки.
3. На третьем этапе необходимо провести объектно-ориентированный анализ задачи и построить архитектуру приложения.
4. На четвертом этапе происходит написание текста программы.
5. После написания программы следует провести тестирование.

# 2 Создание основных компонентов

**Класс «Application»**

На рисунке 1 представлен класс «Application». Данный класс организует работу всего приложения.

|  |
| --- |
| Application |
| - win: SDL\_Window\*  - ren: SDL\_Renderer  - font: TTF\_Font\*  - gameMap: Map\*  - menu: Menu\*  - Running: bool  - Steps: int  - State: char |
| + Exec(): int  - Init(): bool  - Clean(): void  - CheckGameContext(): void  - ExitEvent(): void  -KeyDownEvent(SDL\_Keycode keyCode, Uint16 mod, SDL\_Scancode scancode): void  - drawScore(): void  - OnEvent(SDL\_Event \*event): void |

Рисунок 1 - Класс “Application”

**Поля класса:**

* “win” указатель на класс SDL\_Window, для работы приложения
* “ren” указатель на класс SDL\_Renderer, для рисования на плоскости созданного окна
* “font” объект шрифта для всего приложения
* “gameMap” указатель на объект собственного класса Map, для загрузки/отображения игровой карты.
* “menu” для отрисовки и взаимодействия с пользовательским меню
* “Running” переменная bool, при false приложение заканчивает свою работу
* “Steps” служит для подсчета количества шагов персонажа во время игры
* “State” состояние приложение (отображено главное меню, пауза, игра)

**Методы класса:**

**-** “Exec()” запускает приложение

- “Init()” инициализирует все нужные модули

- “Clean()” служит для очистки ресурсов после завершения цикла приложения

- “CheckGameContext()” проверяет, находятся ли все ящики на своих местах

- “ExitEvent()” присваивает переменной Running значение false

- “KeyDownEvent(SDL\_Keycode keyCode, Uint16 mod, SDL\_Scancode scancode)” функция обработки нажатия клавиш

- “drawScore()” функция рисует количество сделанных шагов

- “OnEvent(SDL\_Event \*event )” основная функция обработки событий

**Класс «Map»**

На рисунке 2 представлен класс «Map». Данный класс нужен для загрузки и отображения игрового поля.

|  |
| --- |
| Map |
| - static\_objects: vector<Object\*>  - boxes: vector<Box\*>  - box\_places: vector<Object\*>  - player: Player\* |
| + loadMapFromFile(): int  + canPlayerMoveLeft(): bool  + canPlayerMoveRight(): bool  + canPlayerMoveTop(): bool  + canPlayerMoveDown(): bool  + canBoxMoveLeft(int x, int y): bool  + canBoxMoveRight(int x, int y): bool  + canBoxMoveTop(int x, int y): bool  + canBoxMoveDown(int x, int y): bool  + drawMap(SDL\_Renderer \*ren): void  + repaint(SDL\_Renderer \*ren, int x, int y): void  + repaintPlayerTopDown(SDL\_Renderer \*ren): void  + repaintPlayerLeftRight(SDL\_Renderer \*ren): void |

Рисунок 2 - Класс “Map”

**Поля класса:**

* “static\_objects” вектор с объектами, которые не будут перемещаться по карте
* “boxex” вектор с объектами коробок
* “box\_places” вектор для проверки, стоят ли все коробки по своим местам
* “player” персонаж игры, которым нужно управлять

**Методы класса:**

**-** “loadMapFromFile()” функция заполняет вектора нужными объектами, считывая информацию из файла

- “canPlayerMoveLeft()”, “canPlayerMoveRight()”, “canPlayerMoveTop()”

canPlayerMoveDown()” функции проверки, сможет ли персонаж в текущий момент двинуться в нужную сторону.

- “canBoxMoveLeft(int x, int y)”, “canBoxMoveRight(int x, int y)”, “canBoxMoveTop(int x, int y)”, “canBoxMoveDown(int x, int y)” функции для проверки, сможет ли коробка с координатами x и y двинуться в нужную сторону.

- “drawMap(SDL\_Renderer \*ren)” функция для отрисовки всей карты

- “repaint(SDL\_Renderer \*ren, int x, int y)” функция для отрисовки только блока с координатами (x, y).

- “repaintPlayerTopDown(SDL\_Renderer \*ren)” функция перерисовки после движения персонажа вверх или вниз.

- “repaintPlayerLeftRight(SDL\_Renderer \*ren)” функция перерисовки после движения персонажа вправо или влево.

**Класс «Menu»**

На рисунке 3 представлен класс «Menu». Данный класс нужен для отображения пользовательского меню.

|  |
| --- |
| Menu |
| - point: SDL\_Rect |
| + drawMainMenu(SDL\_Renderer \*ren): void  + drawPause(SDL\_Renderer \*ren): void  + drawGameEnd(SDL\_Renderer \*ren, int score): void  + drawRules(SDL\_Renderer \*ren): void  + drawTopList(SDL\_Renderer \*ren): void  + enterEvent(char state, SDL\_Renderer \*ren): char  + resetPoint(char state): void  + eventTop(char state, SDL\_Renderer \*ren, int score): void  + eventDown(char state, SDL\_Renderer \*ren, int score): void |

Рисунок 3 - Класс “Menu”

**Поля класса:**

**-** “point” структура, хранящая координаты курсора меню

**Методы класса:**

* “drawMainMenu(SDL\_Renderer \*ren)”
* “drawPause(SDL\_Renderer \*ren)” рисует меню паузы
* “drawGameEnd(SDL\_Renderer \*ren, int score)” рисует меню, когда все ящики находятся на своих местах
* “drawRules(SDL\_Renderer \*ren)” рисует меню с правилами игры
* “drawTopList(SDL\_Renderer \*ren)” выводит на экран таблицу лидеров
* “enterEvent(char state, SDL\_Renderer \*ren)” обработка нажатия на клавишу Enter
* “resetPoint(char state)” сброс курсора до нужных координат для определенного меню
* “eventTop(char state, SDL\_Renderer \*ren, int score)” и “+ eventDown(char state, SDL\_Renderer \*ren, int score)” обрабатывают события нажатия стрелок “вверх” и “вниз” соответственно.

**Класс «Object»**

На рисунке 4 представлен класс «Object». Данный класс является родительским классом для всех объектов карты (коробка, игрок, кирпичный блок).

|  |
| --- |
| Object |
| - m\_type: int  - m\_x: int  - m\_y: int |
| + Object(int t, int x, int y)  + getType(): int  + getX(): int  + getY(): int  + setX(int x): void  + setY(int y): void  + onRender(SDL\_Renderer\* ren): void |

Рисунок 4 - Класс “Object”

**Поля класса:**

**-** “m\_type” тип объекта

- “m\_x” координата x объекта

- “m\_y” координата y объекта

**Методы класса:**

**-** “Object(int t, int x, int y)” конструктор класса с параметрами: тип, x, y соответственно.

**-** “getType()”, “getX()”, “getY()”, “setX(int x)”, “setY(int y)” геттеры и сеттеры для полей класса.

- “onRender(SDL\_Render \*ren)” виртуальная функция для отрисовки объекта, которая будет перегружена во всех потомках данного класса

**Статические классы**

На рисунке 5 представлено описание классов (Brick, Desert, BoxPlace, Grass), унаследованных от «Object». Структура у всех одинаковая, отличается лишь картинка для отображения.

|  |
| --- |
| Static classes |
| - image: SDL\_Surface\* |
| + onRender(SDL\_Renderer \*ren): void |

Рисунок 5 – Статические классы

**Поля класса:**

* “image” хранит в себе изображение данного объекта

**Методы класса:**

- “onRender(SDL\_Renderer \*ren) ” перегруженная функция для вывода объекта на экран

**Динамические классы**

На рисунке 6 представлено описание классов (Player и Box), унаследованных от «Object». Они, в отличие от предыдущих, могут перемещаться по карте.

|  |
| --- |
| Dynamic classes |
| - image: SDL\_Surface\* |
| + onRender(SDL\_Renderer \*ren): void  + onMove(int dx, int dy): void |

Рисунок 6 – Динамические классы

**Поля класса:**

* “image” хранит в себе изображение данного объекта

**Методы класса:**

- “onRender(SDL\_Renderer \*ren) ” перегруженная функция для вывода объекта на экран

- “onMove(int dx, int dy)” функция для изменения координат объекта (движение его по карте)

# 3 Организация программы

На рисунке 7 представлена диаграмма UML.

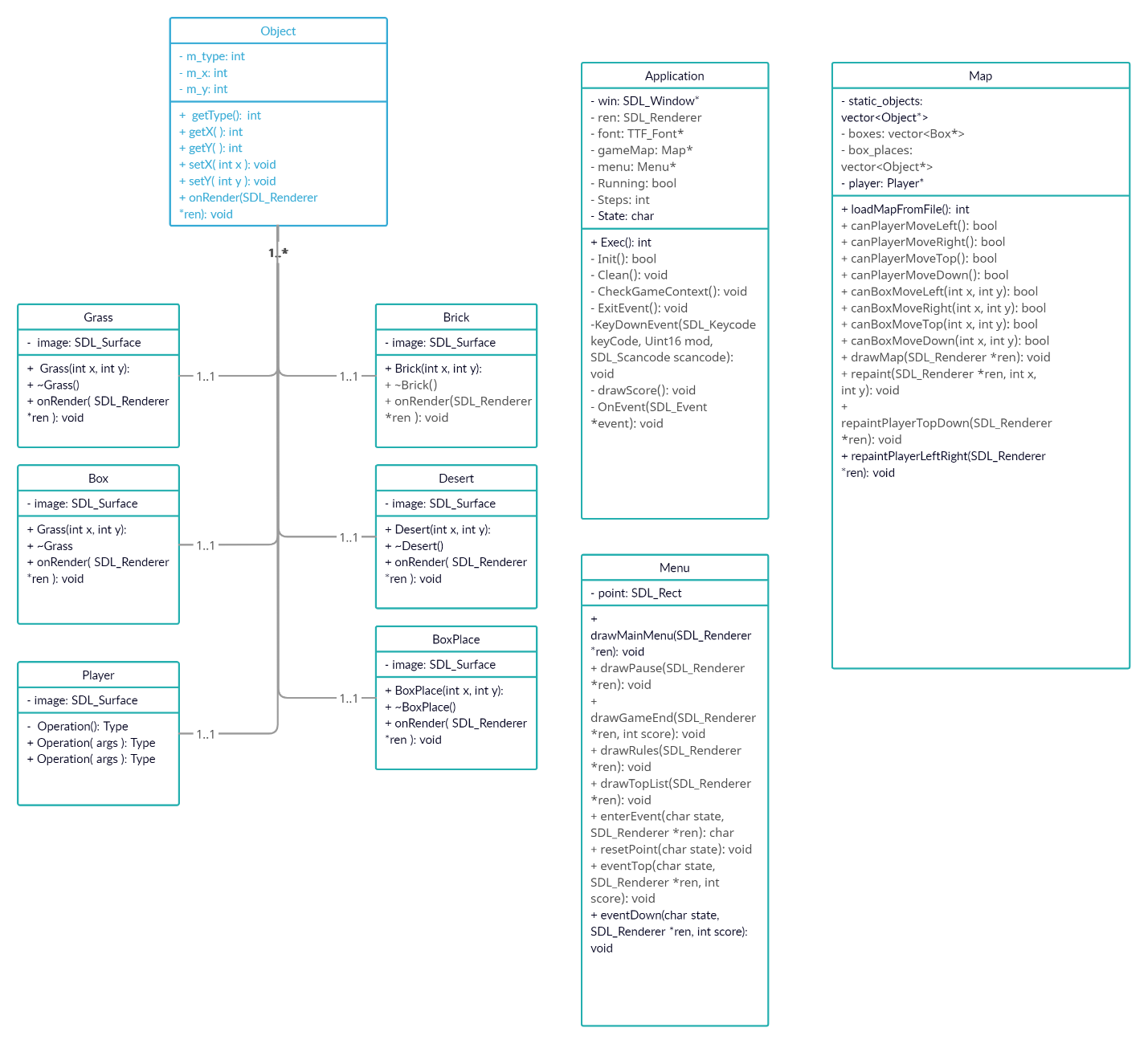


Рисунок 7 – Диаграмма UML

# ЗАКЛЮЧЕНИЕ

В ходе работы было решено во время игрового процесса перерисовывать не все блоки, находящиеся на карте, а только несколько блоков вокруг персонажа. Это значительно повысило производительность приложения.

Результаты работы удовлетворяют поставленной задаче. Считаю, что цель курсовой работы достигнута.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Библиотека SDL2 URL: https://wiki.libsdl.org/ (дата обращения: 07.12.2020)
2. Галовиц Я. Г. С++17 STL. Стандартная библиотека шаблонов. — СПб.: Питер, 2018. — 432 с.

# ПРИЛОЖЕНИЕ

Приложение содержит в себе исходные файлы проекта.

Имя файла: «Приложение.zip»

Для разработки программы используется среда программирования CodeBlocks.

Для написания программы подключены основная библиотека SDL и дополнительные библиотека для работы с текстом SDL\_ttf.

Проект включает в себя файлы:

* + - 1. Application.h
      2. Box.bmp
      3. Box.h
      4. Box\_place.bmp
      5. BoxPlace.h
      6. Brick.bmp
      7. Brick.h
      8. Desert.bmp
      9. Desert.h
      10. GameEnd.bmp
      11. Grass.bmp
      12. Grass.h
      13. Main.cpp
      14. MainMenu.bmp
      15. Map.h
      16. Map.txt
      17. Menu.h
      18. Object.h
      19. Pause.bmp
      20. Player.bmp
      21. Player.h
      22. Records.txt
      23. Rules.bmp
      24. TopList.bmp