Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

Приложение «игра Арканоид»

по дисциплине

«Конструирование программ и языки программирования»

КП БГУИР 1-400201.412 ПЗ

Выполнил:
Студент гр. 050504
Матусевич С. К.

Руководитель:
Байдун Д. Р.

Минск 2021

СОДЕРЖАНИЕ

ЗАДАНИЕ

по курсовому проектированию студента
Матусевич Семёна Климентьевича

1 Тема проекта «Игра Арканоид»
2 Срок сдачи студентом законченного проекта: 26 декабря 2021 г.
3 Исходные данные к проекту *Операционная система Linux и Windows, использование языка C++ и IDE Qt Creator.*
4 Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке):

Введение.
1. Обзор литературы. 2. Перечень используемых сокращений. 3. Обзор методов и алгоритмов решения. 4. Обоснование выбранных методов и алгоритмов. 5. Обзор методов и алгоритмов решения. 6. Описание алгоритмов решения задачи. 7. Руководство пользователя.
Заключение
Список литературы.

5 Перечень графического материала (с точным обозначением обязательных чертежей и графиков):

1. Листинг программы
2. Диаграмма классов
3. Скриншоты работающей программы
4. Блок-схема алгоритма

6 Консультант по проекту (с обозначением разделов проекта) *Байдун Д.Р.*
7 Дата выдачи задания *14 сентября 2021 г.*
8 Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

разделы 1,2 к 05.10. – 10 %;
раздел 3 к 25.10. – 15 %;

раздел 4 к 07.11. – 15 %;
разделы 5,6 к 19.11. – 25 %;
раздел 7 к 26.11. – 25 %;
оформление пояснительной записки и графического материала к 01.12.2021 – 10 %.
Защита курсового проекта 24.12.2021 г.

РУКОВОДИТЕЛЬ _____ *Д.Р. Байдун*
(подпись)

Задание принял к исполнению _____ *С.К. Матусевич*
(дата и подпись студента)

ВВЕДЕНИЕ

Сегодня игра “Аракноид” очень распространена, хотя пиком их популярности были девяностые и двухтысячные. Я всегда любил компьютерные игры и решил попробовать создать свою игру. Выбрал жанр платформер так как мне нравятся игры данного жанра, и они не слишком сложны в реализации. Потенциал для развития этого проекта я считаю почти бесконечным так как можно придумывать новые уровни, рассказывать истории с помощью игры и улучшать графическую реализацию.

Объектно-ориентированное программирование представляет собой технологию программирования, которая базируется на классификации и абстракции объектов. Одним из наиболее популярных средств объектно-ориентированного программирования, позволяющим разрабатывать программы, эффективные по объёму кода и скорости выполнения является C++.

C++ — компилируемый, статически типизируемый язык общего назначения. Он поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает в себе как возможности низкоуровневых языков программирования, так и возможности высокоуровневых.

Основными концепциями объектно-ориентированного программирования являются инкапсуляция, полиморфизм и наследование. Язык C++ предоставляет исчерпывающие возможности для реализаций этих концепций. В результате использования инкапсуляции, программа, написанные на C++, обладает повышенной защищённостью объектов от влияния на них кода других частей этой же программы. Наследование предоставляет важную возможность повторного использования кода, что может значительно уменьшить количество кода, однако требует предварительного проектирования архитектуры. Полиморфизм позволяет программе быть более гибкой. Такая система удобна в тестировании и позволяет легко заменять и модифицировать свои компоненты.

Универсальность и гибкость языка позволяют использовать его в различных целях. Область применения данного языка включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр). Исходя из этого можно считать, что данный язык достаточно удобен для написания выбранной курсовой работы.

1 ОБЗОР ИСТОЧНИКОВ

1.1 Анализ аналогов программного средства

Arkanoid – аркадная видеоигра для игровых автоматов, разработанная компанией Taito в 1986 году. Игра основана на играх серии Breakout фирмы Atari. Именно её название стало нарицательным для класса подобных игр.

Игрок контролирует небольшую платформу-ракетку, которую можно передвигать горизонтально от одной стенки до другой, подставляя её под шарик, предотвращая его падение вниз. Удар шарика по кирпичу приводит к разрушению кирпича.

После того как все кирпичи на данном уровне уничтожены, происходит переход на следующий уровень, с новым набором кирпичей. Есть и некоторое разнообразие: определенные кирпичи нужно ударять несколько раз.

Было создано четыре версии игрового автомата: Arkanoid (1986), Tournament Arkanoid (1986), Revenge of Doh (1987, Arkanoid *II*), а также Arkanoid Returns (1997).

В середине-конце 1980-х годов версии игры выходили для различных 8-разрядных бытовых компьютеров — (Amstrad CPC, Apple II, Atari 800, Commodore 64, MSX, ZX Spectrum), игровых консолей (Game Boy, NES), и 16-разрядных компьютеров (Amiga, Apple IIgs, Atari ST).

Также существовали неофициальные версии для советских бытовых компьютеров, например, для БК-0010 и Вектор-06Ц. Версия для игровой консоли SNES называлась Arkanoid: Doh It Again (1997).

В Японии для Sony Playstation была выпущена версия Arkanoid Returns и сиквел Arkanoid Returns *2000*.

1.2 Постановка задачи

Для реализации игры будем использовать фреймворк Qt.

Это очень удобный конструктор графической оболочки, который так же имеет свой встроенный методы для разработки кроссплатформенных приложений, отличное решение для Арканоида.

В программе будут реализованы функции:

- Пауза и продолжение игры.
- Разрушение блоков под действием удара
- Звуковые эффекты и сопровождение
- Основная система будет Linux.

Для реализации данного программного обеспечения используется объектно-ориентированный язык программирования C++.
Разработка производилась в Qt Creator[2] версии 5.12.12.

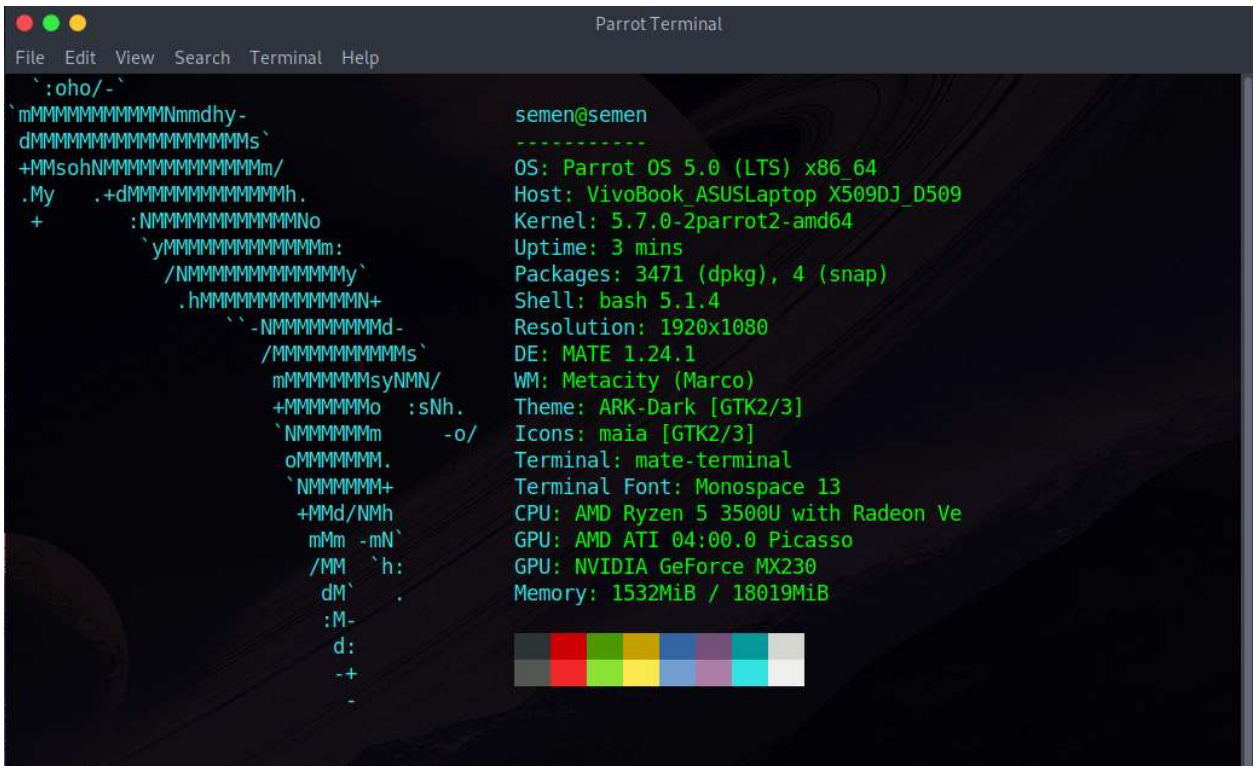

Рис 1.1 - Информация о системе.

2 ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ

Для начала надо определить одни из самых важных определений при создании игры. Это paddle, rect.

- Rect (от английского слова rectangle) – это прямоугольник вокруг какого-то объекта. Он нужен чтобы мы могли управлять этим объектом и создавать логику взаимодействий предметов.
- Paddle – Наша ракета (платформа), на которую падает шарик.

-Multimedia – библиотека для работы со звуком.

3 ОБЗОР МЕТОДОВ И АЛГОРИТМОВ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

Основной алгоритм множества игр — это цикл в котором мы обрабатываем разные действия пользователя (например, нажатие кнопок), обновляем состояния разных объектов (например, перемещение главного персонажа) и в конце после этого происходит отрисовка новых позиций объектов. Программа проходит по этому циклу очень большое количество раз, и мы многих итераций цикла просто не замечаем. Это сделано для более плавной картинки и удобства работы.

Для создания окна в котором будет отображаться игра мы используем класс MainWindow [2] в конструкторе, которого мы задаём размер окна в пикселях и название этого окна.

В программе были созданы 4 класса: Breakout, Paddle, Brick, Ball.

- Класс Paddle описывает игрока (платформу). У класса Paddle есть поля dx это его скорость по координатам x, он может перемещаться в лево и в право, rect- это его положение на поле и его размер, image– это переменная, которая отвечает за картинку самой “ракетки”. Метод move() перемещает прямоугольник с изображением Ракетки. Направление движения задается переменной dx. Метод resetState() устанавливает Ракетку в исходное положение.
- Класс Brick описывает блоки. У класса Brick такие же поля и геттеры/сеттеры, как и у Paddle, но добавляется одна булевская переменная destroyed, для доступа к ней создан метод isDestroyed(), который показывает разрушен ли данный блок или нет, в зависимости от этого, отображает его в окне. Конструктор класса Brick загружает изображение Кирпича, иницилизирует переменную-флаг destroyed и устанавливает изображение в исходную позицию:
- Класс Ball описывает наш мяч. В переменных xdir и ydir хранится направление движения Мяча. В начале игры Мяч движется в направлении вправо-вверх: xdir = -1, ydir = -1. Метод autoMove() вызывается в каждом игровом цикле для перемещения Мяча по экрану. Если Мяч достигает границ окна (за исключением нижней), то он меняет свое направление. Если же Мяч попадает в нижнюю границу окна, то назад он не отскакивает, а игра при этом считается завершенной.
- Класс Breakout хранит состояние нашей игры.

Ракетка управляется с помощью клавиш-стрелок на клавиатуре. В игре мы следим за событиями нажатия клавиш клавиатуры с помощью следующих методов: keyPressEvent, keyReleaseEvent

В переменной x хранится текущее положение Ракетки по оси x. Переменная timerId используется для идентификации объекта timer. Это необходимо в тех моментах, когда мы приостанавливаем игру, Константа N_OF_BRICKS задает количество Кирпичей в игре, в нашем случае их 30. Константа DELAY управляет скоростью игры, также присутствуют

переменные-указатели на объекты Мяча, Ракетки и массива Кирпичей. Булевские переменные : gameOver, gameStarted, gameWon, paused, отвечают за состояния игры. Метод drawObjects() отрисовывает в окне все объекты игры: Мяч, Ракетку и Кирпичи. А так как данные объекты представлены изображениями, то при помощи метода drawImage() мы отображаем и их изображения. Метод finishGame() отображает завершающее сообщение в центре окна. Это либо "Game lost" ("Игра проиграна"), либо "Victory" ("Победа"), метод moveObjects вызывает у объектов шара и ракетки методы движения. Метод checkCollision орагнизовывает столкновение наших объектов друг с другом, изменяя при этом скорость и направление шарика. В теле метода timerEvent() мы сначала перемещаем объекты, а затем проверяем, не столкнулся ли Мяч с Ракеткой или Кирпичом. В конце генерируем событие отрисовки. Метод startGame() сбрасывает состояния объектов ball и paddle; они перемещаются в исходное положение.

Дальше для запуска игры мы должны нажать пробел и наш шарик полетит в сторону блоков. В игре используется три цвета кирпича: синий, красный и жёлтый. У самого первого 3 жизни, а у остальных на единицу меньше. Такое разнообразие реализовано за счёт 3-х конструкторов с различным числом аргументов. При каждом новом запуске игры, будет сгенерирована новая раскладка кирпичей. Это вызвано тем, что каждый раз, при создании кирпича, его конструктор определяется случайным образом. Для того, чтобы сгенерированные числа были всегда разные, использовали переменную QRandomGenerator *rg = QRandomGenerator::global(). При попадании шарика по блоку или стенке метод checkCollission проверяет вышел ли за нижнюю границу шарик, если да то игра завершается с надписью “Lost”, иначе проверяем количество разрушенных Кирпичей. Если все Кирпичи уничтожены, то мы выиграли. При столкновении шарика и платформы, шарик должен отлететь вверх в какую-то сторону.

Если мяч ударяется о нижнюю часть Кирпича, то мы меняем направление у мяча — он идет вниз. Количество жизней кирпича хранится в массиве colors. При попадании шарика на синий кирпич, он становится красным. При попадании шарика на красный кирпич, он становится желтым. При попадании шарика на желтый кирпич, он разрушается. Какой кирпич нужно заменить на новый, определяется по координатам, которые записаны в массив coord.

С помощью метода keyPressEvent(QKeyEvent *e) мы понимаем какую клавишу мы нажали и что нам в этом случае надо делать, например при нажатии на ESC программа завершается посредством метода exit, стрелочки отвечают за движение, а клавиша P за паузу.

Для улучшения пользовательского интерфейса, автором было принято решение добавить фоновый звук, задний фон и различные звуки для победы/проигрыша. Фоновый звук добавляем при помощи метода play, библиотеки Multimedia, который принимает путь до нашего wav файла, в нашем случае он хранится в папке music. В случае победы или проигрыша

добавляем соответствующий звук “win.wav” or “lose.wav”. Также автор добавил фоновое изображение посредством класса QPallete и его методом setBrush.

4 ОБОСНОВАНИЕ ВЫБРАННЫХ МЕТОДОВ И АЛГОРИТМОВ

Qt позволяет запускать написанное с его помощью программное обеспечение в большинстве современных операционных систем путём простой компиляции программы для каждой системы без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Является полностью объектно-ориентированным, расширяемым и поддерживающим технику компонентного программирования.

Создание класса Paddle, Ball, Brick позволяет нам более удобно управлять и создавать объекты. Мы описали почти всю логику объектов в этих классах.

С помощью массива colors мы узнаем значение жизни каждого блока, однако если б мы использовали другую структуру данных, то доступ происходил бы быстрее.

Когда мы обрабатываем столкновения шарика с ракеткой, в зависимости от зоны, которых 4, шарик ведёт себя по-разному. Но такой подход имеет недостаток в виде маленького числа разбиений платформы и из-за этого спустя пару игр возможно предсказывать полёт шарика, что не в лучшую сторону сказывается на игре.

5 ОПИСАНИЕ ПРОГРАММЫ ДЛЯ ПРОГРАММИСТА

Диаграмма классов приведена в Приложении В на чертеже ГУИР. 400201.412 РР

Диаграмма классов позволяет нам увидеть зависимости, которые есть между классами в программе.

6 ОПИСАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ

6.1 Алгоритм перерисовки объектов. Метод void Breakout::paintEvent(QPaintEvent *e)

Шаг 1. Создание объекта класса QPainter: QPainter painter (this)
Шаг 2. Если gameOver равно значению true, то Шаг 3, иначе Шаг 5
Шаг 3. Проигрываем звук поражения: QSound::play(":/music/lose.wav");
Шаг 4. Переходим в функцию отрисовски сообщения о поражении finishGame(&painter, "Game lost")
Шаг 5. Если gameWon равно значению true, то Шаг 6, иначе Шаг 8
Шаг 6. Проигрываем звук победы: QSound::play(":/music/win.wav")
Шаг 7. Переходим в функцию отрисовски сообщения о победе finishGame(&painter, "Victory")
Шаг 8. Продолжаем отрисовку объектов : drawObjects(&painter)
Шаг 9. Конец

6.2 Алгоритм рисования блоков. Метод void Breakout::drawObjects(QPainter *painter))

Шаг 1. Вызываем метод drawImage(ball->getRect(),ball->getImage()) у painter
Шаг 2. Вызываем метод drawImage(ball->getRect(),paddle->getImage()) у painter
Шаг 3. Цикл for int i=0; i < N_OF_BRICKS; i++
Шаг 4. Если блок не разрушен: то вызываем метод у painter, painter ->drawImage(bricks[i]->getRect(), bricks[i]->getImage()) Иначе Шаг 5
Шаг 5. Конец цикла с шага 3
Шаг 6. Конец

6.3 Алгоритм столкновений игрока с блоками. Метод void Breakout::checkCollision()

Шаг 1. Если ball -> getRect().bottom() больше BOTTOM_EDGE то вызываем метод void Breakout::stopGame()
Шаг 2. Если (ball -> getRect(). bottom() меньше 0, то Шаг 3, иначе Шаг 7
Шаг 3. Присваиваем указателю типа QRandomGenerator значение QRandomGenerator::global()
Шаг 4. Создаём переменную e типа int и присваиваем ей значение rg -> bounded(1, 10)

Шаг 5. Если переменная e чётная ball -> setXDir(-1) – шарик оттолкнётся влево , иначе ball -> setXDir(1) – шарик оттолкнётся вправо

Шаг 6. ball -> setYDir(+2) – устанавливаем скорость шарика по оси Y вниз

Шаг 7. Цикл for int i=0, j=0; i<N_OF_BRICKS; i++

Шаг 8. Если brick[i] ->isDestroyed() вернул true, то j++

Шаг 9. Если переменная j равна числу блоков (N_OF_BRICKS) вызываем метод void Breakout: victory ()

Шаг 10. Конец цикла с шага 7

Шаг 11. Если мячик касается платформы: (ball->getRect()).intersects(paddle->getRect()) равно true, то Шаг 12 иначе Шаг 33

Шаг 12. Создаем переменную paddleLPos типа int равную paddle->getRect().left()

Шаг 13. Создаем переменную ballLPos типа int равную ball->getRect().left()

Шаг 14. Создаем переменную first типа int равную paddleLPos + 3

Шаг 15. Создаем переменную second типа int равную paddleLPos + 25

Шаг 16. Создаем переменную third типа int равную paddleLPos + 50

Шаг 17. Создаем переменную fourth типа int равную paddleLPos + 75

Шаг 18. Если ballLPos меньше first Шаг 19 иначе Шаг 21

Шаг 19. Ставим направление шарика по иксам в лево ball->setXDir(-1)

Шаг 20. Ставим направление шарика по игрекам в верх ball->setYDir(-2)

Шаг 21. Если ballLPos больше либо равен first и ballLPos меньше second Шаг 22 иначе Шаг 24

Шаг 22. Ставим направление шарика по иксам в лево ball->setXDir(-1)

Шаг 23. Ставим направление шарика по игрекам в противоположную сторону относительно его движения ball->setYDir(-1*ball->getYDir())

Шаг 24. Если ballLPos больше либо равен second и ballLPos меньше third (средняя часть ракетки) Шаг 25 иначе Шаг 27

Шаг 25. Ставим направление шарика по иксам в 0, т.е он двигается только по игрекам ball->setXDir(0)

Шаг 26. Ставим направление шарика по игрекам ball->setYDir(-3)

Шаг 27. Если ballLPos больше либо равен third и ballLPos меньше fourth Шаг 28 иначе Шаг 30

Шаг 28. Ставим направление шарика по иксам в право ball->setXDir(1)

Шаг 29. Ставим направление шарика по игрекам в противоположную сторону относительно его движения ball->setYDir(-1*ball->getYDir())

Шаг 30. Если ballLPos больше fourth Шаг 31, иначе Шаг 33

Шаг 31. Ставим направление шарика по иксам в право ball->setXDir(1)

Шаг 32. Ставим направление шарика по игрекам в верх ball->setYDir(-2)

Шаг 33. Цикл for int i=0; i < N_OF_BRICKS; i++

Шаг 34. Если (ball->getRect()).intersects(bricks[i]->getRect()) равна true Шаг 35, иначе Шаг 64

Шаг 35. Создаем переменную ballLeft типа int равную ball->getRect().left()

Шаг 36. Создаем переменную ballHeight типа int равную ball->getRect().height()

Шаг 37. Создаем переменную ballWidth типа int равную ball->getRect().width()

Шаг 38. Создаем переменную ballTop типа int равную ball->getRect().top()

Шаг 39. Создаем объект pointRight типа QPoint равную pointRight(ballLeft + ballWidth + 1, ballTop)

Шаг 40. Создаем объект pointLeft типа QPoint равную pointLeft(ballLeft - 1, ballTop)

Шаг 41. Создаем объект pointTop типа QPoint равную pointTop(ballLeft, ballTop -1)

Шаг 42. Создаем объект pointBottom типа QPoint равную pointBottom(ballLeft, ballTop + ballHeight + 1)

Шаг 43. Если i-тый блок не разрушен то Шаг 44, иначе Шаг 64

Шаг 44. Если i-тый блок содержит правую точку шарика, то ставим его направление в лево: bricks[i]->getRect().contains(pointRight) Шаг 45 иначе Шаг 46

Шаг 45. ball->setXDir(-1)

Шаг 46. Если i-тый блок содержит левую точку шарика, то ставим его направление в право: bricks[i]->getRect().contains(pointLeft) Шаг 47 иначе Шаг 48

Шаг 47. ball->setXDir(1)

Шаг 48. Если i-тый блок содержит верхнюю точку шарика то ставим его направление в низ: bricks[i]->getRect().contains(pointTop) Шаг 49 иначе Шаг 50

Шаг 49. ball->setYDir(2)

Шаг 50. Если i-тый блок содержит нижнюю точку шарика то ставим его направление в верх: bricks[i]->getRect().contains(pointBottom) Шаг 51 иначе Шаг 52

Шаг 51. ball->setYDir(-2)

Шаг 52. Если i-тый элемент массива colors равен 0 то Шаг 53 иначе Шаг 54

Шаг 53. Ставим значение переменной destroyed true : bricks[i]->setDestroyed(true)
Шаг 54. Если i-тый элемент массива colors равен 1 то Шаг 55 иначе Шаг 59
Шаг 55. Создаем переменную w типа int равную coord[i]/10
Шаг 56. Создаем переменную h типа int равную coord[i]%10
Шаг 57. Создаём объект класса Brick: bricks[i] = new Brick (h * 90+60, w * 30+100)
Шаг 58. Устанавливаем значение i-той переменной массива colors нулём : colors[i] = 0
Шаг 59. Если i-тый элемент массива colors равен 2 то Шаг 60 иначе Шаг 64
Шаг 60. Создаем переменную w типа int равную coord[i]/10
Шаг 61. Создаем переменную h типа int равную coord[i]%10
Шаг 62. Создаём объект класса Brick: bricks[i] = new Brick (h*90+60, w*30+100, 1)
Шаг 63. Устанавливаем значение i-той переменной массива colors единицей : colors[i] = 1
Шаг 64. Конец цикла с шага 33
Шаг 65. Конец

6.4 Схема алгоритма функции void Breakout::keyPressEvent(QKeyEvent *e)

Схема алгоритма представлена на чертеже ГУИР.400201.412 Г.1

Данный метод предназначен для обработки событий нажатия клавиш, относящиеся к нашей игре

6.5 Схема алгоритма функции void Breakout::keyReleaseEvent(QKeyEvent *e)

Схема алгоритма представлена на чертеже ГУИР.400201.412 Г.2

Данный метод предназначен для обработки событий отпускания клавиш движения ракетки.

7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Программа состоит из 4 файлов с кодом:

- main.cpp – в этом файле располагается главная функция main;
- breakout.cpp – в этом файле располагается определение методов класса Breakout;
- breakout.h – в этом файле располагается объявление методов и переменных класса Breakout;
- paddle.cpp – в этом файле располагается определение методов класса Paddle;
- paddle.h – в этом файле располагается объявление методов и переменных класса Paddle;
- brick.cpp – в этом файле располагается определение методов класса Brick;
- brick.h – в этом файле располагается объявление методов и переменных класса Brick;
- ball.cpp – в этом файле располагается определение методов класса Ball;
- ball.h – в этом файле располагается объявление методов и переменных класса Brick.

Также в программе используются изображения в формате png. Они лежат в папке img, а звуки в папке music.

Для подключения библиотеки Qt нужно скачать файл с официального сайта [2] библиотеку собрать её и добавить, как third-party library. Но более простой способ — это просто скачать Qt версии 5, т.к в 6 ещё не завезли поддержку библиотек Multimedia, и возможно к концу этого года будет вполне stable версия. Проект собирался с помощью утилиты CQtDeployer[3], были созданы установщики на Windows и Linux, также создан был deb пакет вместе с архивами.

Клавиши управления:

- Стрелочка влево – передвижение влево;
- Стрелочка вправо – передвижение вправо;
- Клавиша P – пауза игры;
- Escape – выход из игры.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данного курсового проекта было изучено и успешно использовано большое количество текстовой и графической информации, в результате чего стало возможным проектирование программного продукта, его тестирование и устранение ошибок.

Данный проект может быть усовершенствован в следующих направлениях:

- Добавление новых механик (например, различные мини-игры и мутаторы для ракетки)
- Улучшение графических возможностей игры
- Добавление редактора уровней
- Добавление смены скинов у шарика

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Страуструп, Б. Язык программирования C++/ Б.Страуструп; специальное издание. Пер. с англ. – Спб.: BHV, 2008. – 1098 с.

[2] Официальный сайт документации Qt [Электронный ресурс].-2021-Режим доступа - https://doc.qt.io/qt-5/ - Дата доступа – 10.12.2021

[3] Официальная документация CQtDeployer [Электронный ресурс].-2021-Режим доступа - https://github.com/QuasarApp/CQtDeployer - Дата доступа – 10.12.2021

ПРИЛОЖЕНИЕ А
(*обязательное*)

Листинг программы с комментариями

ПРИЛОЖЕНИЕ Б
(*обязательное*)

Скриншоты работы программы

На этапе тестирование проверяется правильность выполнения основных действий, совершаемых в ходе игры. Игровое окно на Windows и Linux (рис.

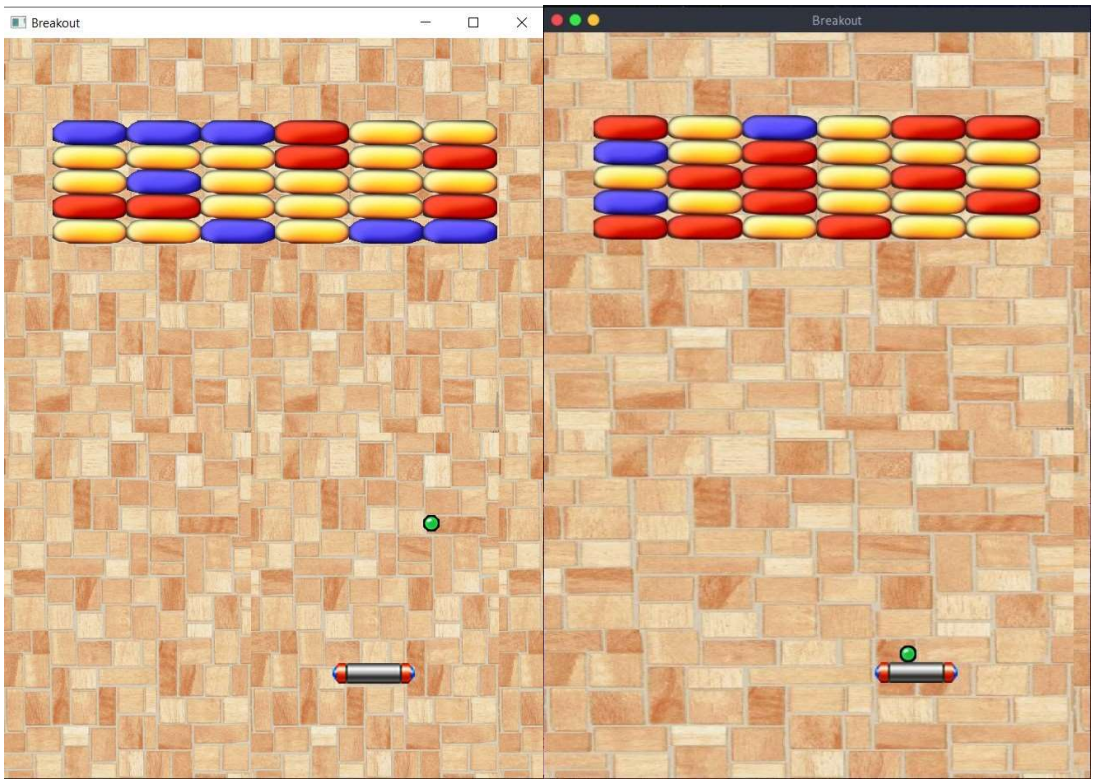

Рис 6.1 – Игровые окна на двух системах

При попадании на кирпич, шарик отлетает в зависимости от того, на какую сторону кирпича он попал (рис 6.2).

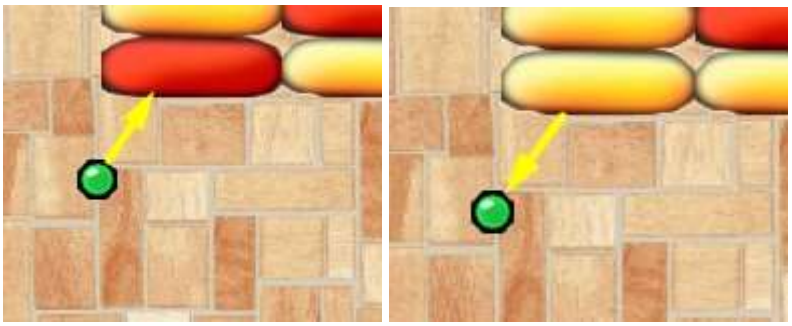

Рис 6.2 – Попадание шарика на кирпич

В зависимости от попадания шарика на конкретную часть платформы, происходит различное изменение скоростей и направлений. Лева часть платформы(рис 6.3)

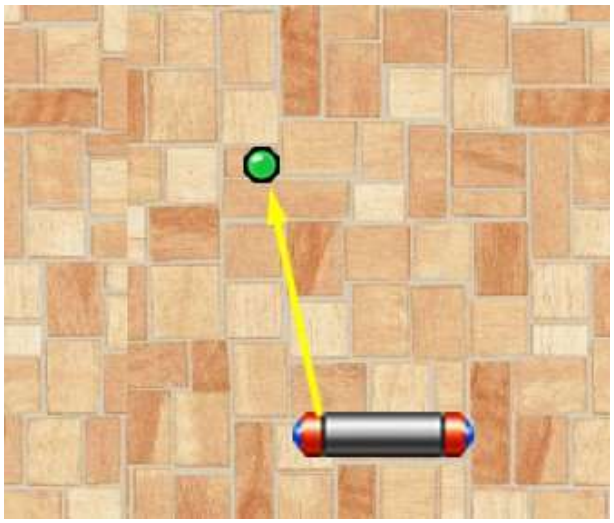

Рис 6.3 – Левая часть платформы

Средняя часть платформы (рис 6.4)

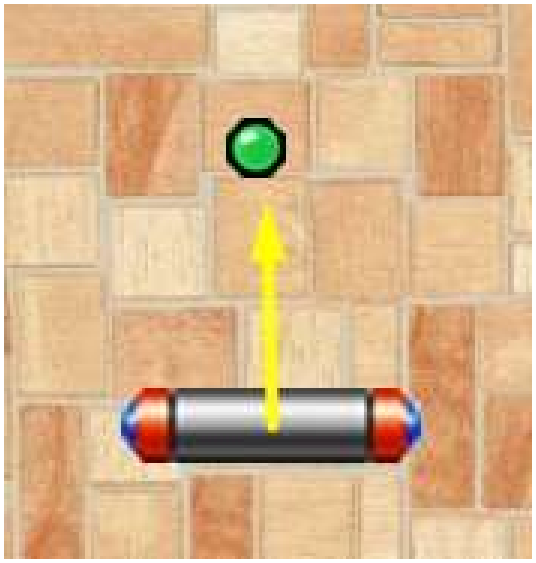

Рис 6.4 – Средняя часть платформы

Правая часть платформы (рис 6.5)

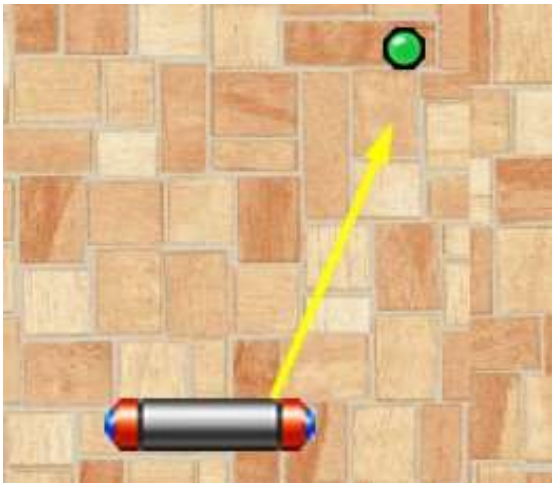

Рис 6.5 – Правая часть платформы

При попадании на кирпич, шарик отлетает в зависимости от того, на какую сторону кирпича он попал.

Попадание шарика на левую сторону кирпича (рис 6.6)

Попадание шарика на правую сторону кирпича (рис 6.7)

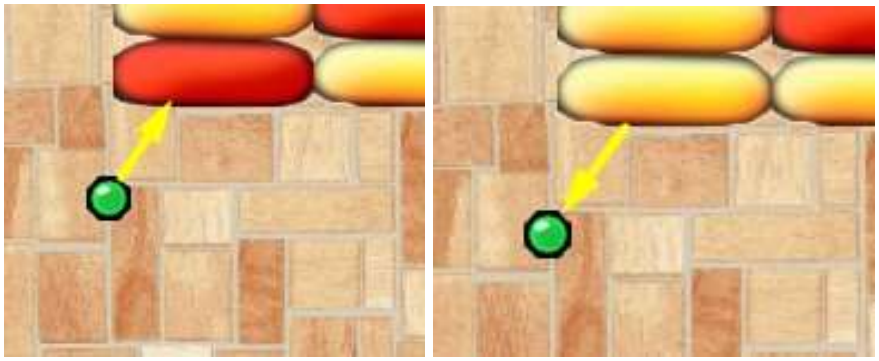

Рис 6.6 – Попадание шарика на левую сторону

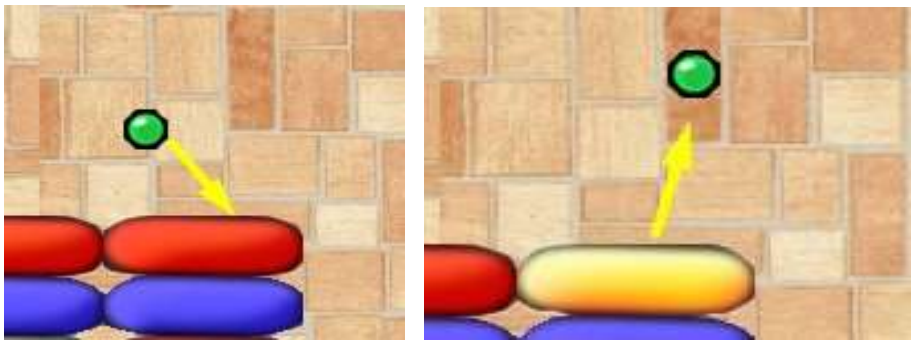

Рис 6.7 – Попадание шарика на левую сторону

При попадании на правую/левую границу, шарик отлетает в противоположном направлении (рис. 6.8)

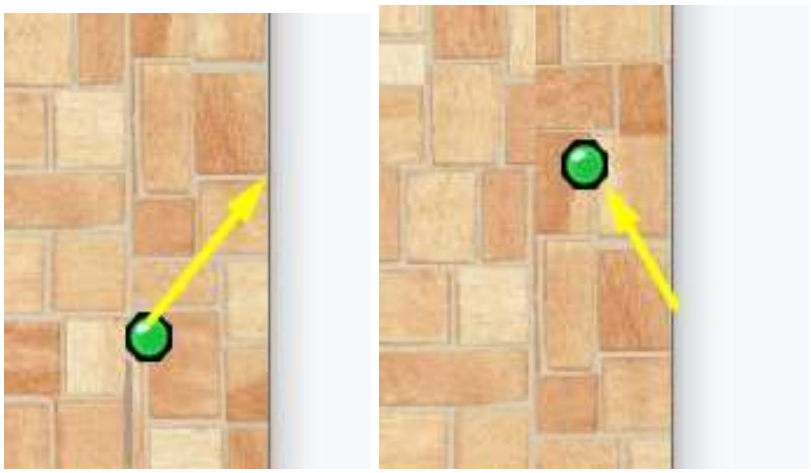

Рисунок 6.8 – Попадание мячика на границу игровой области

Как выглядит установщик созданный CQtDeployer[3] (рис 6.9)

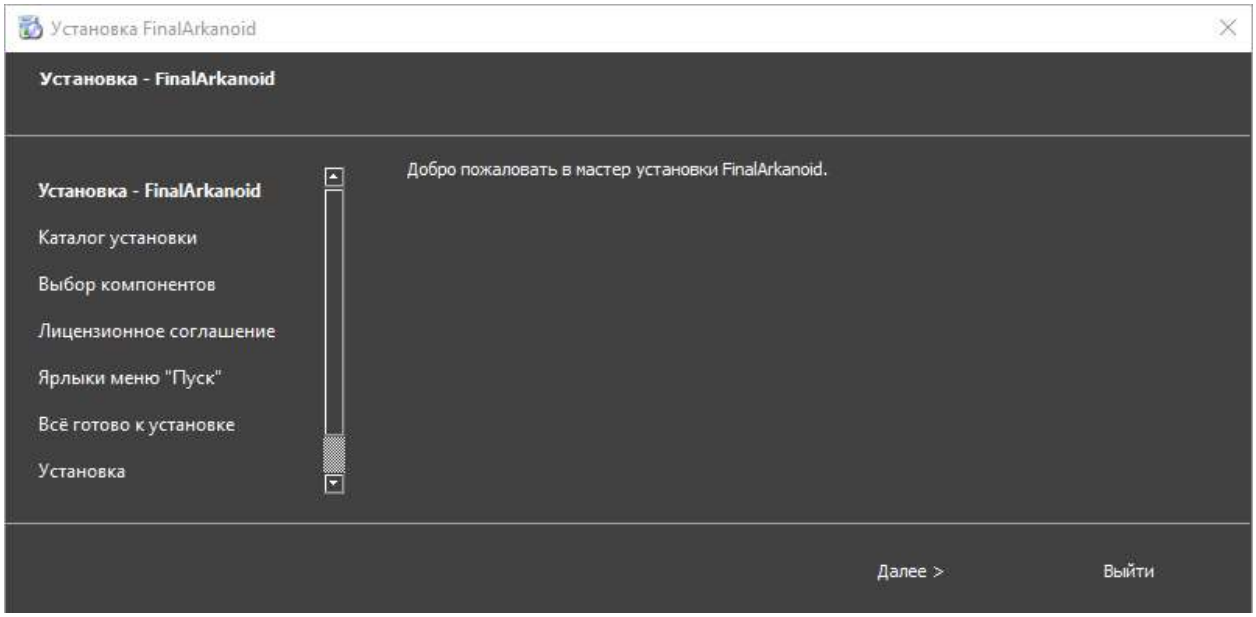

Рис 6.9 – Окно установки программы

Примеры расположения кирпичей (рис 6.10)

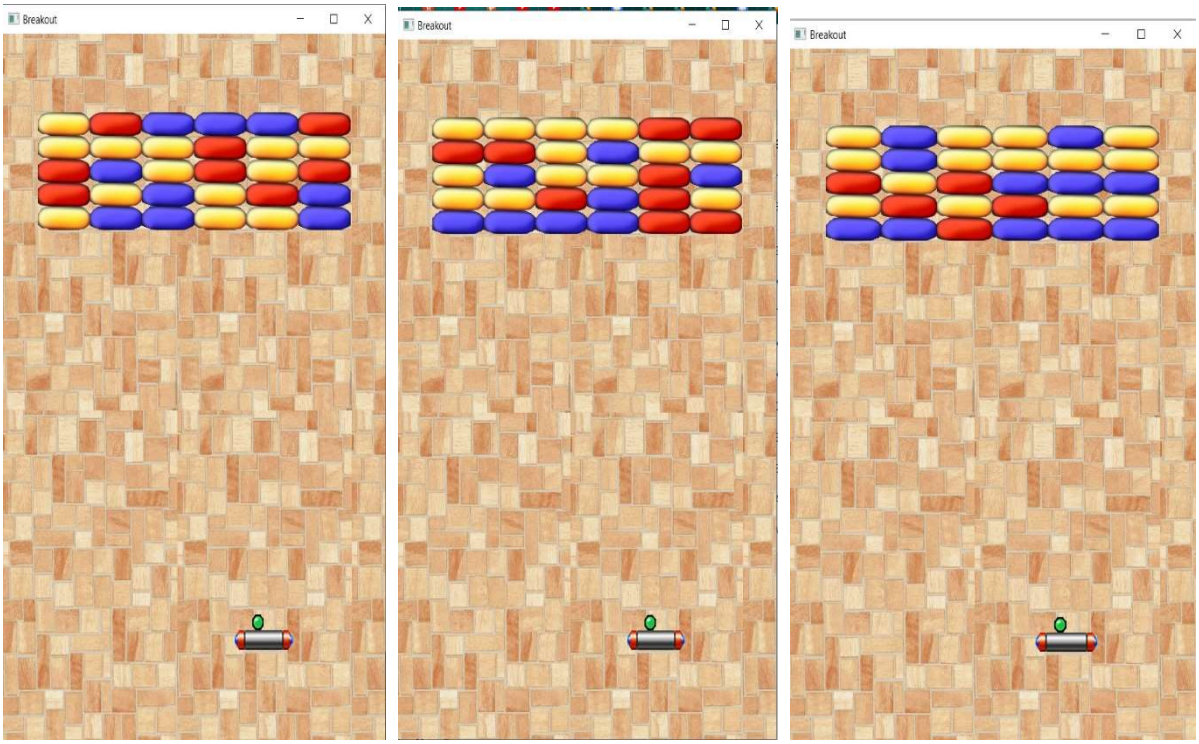

Рис. 6.10 – Пример различной генерации кирпичей

Пауза игры (да,это скриншот), но просто поверьте мне (рис 6.11)

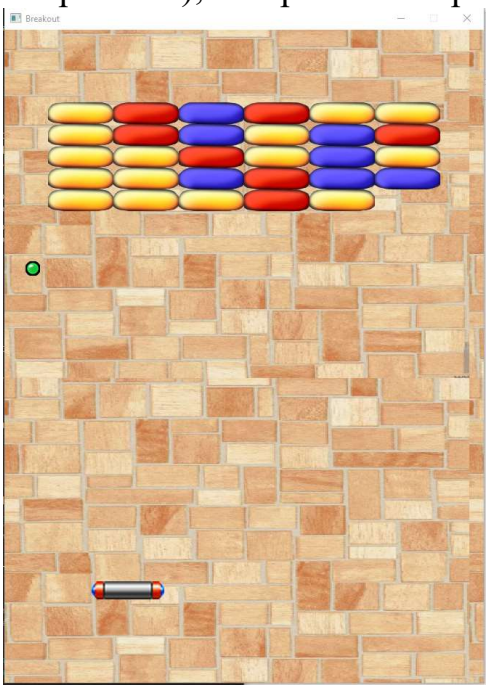

Рис. 6.11 – Пауза игры

Сообщение от игры при победе или поражении (рис 6.12)

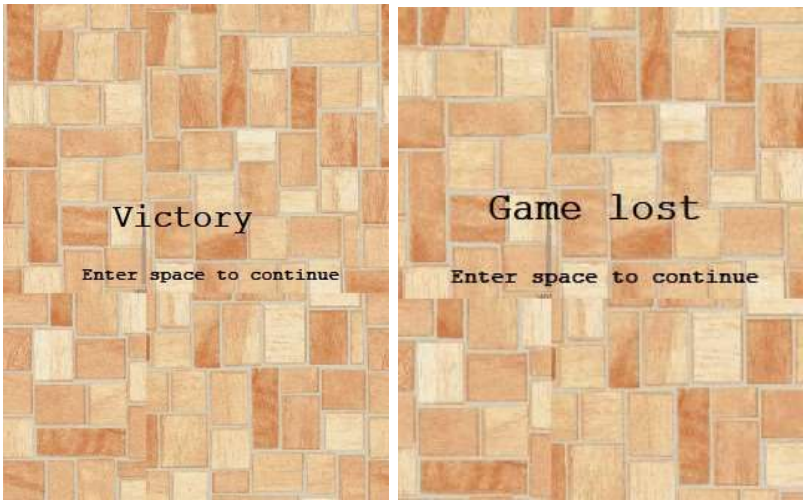

Рисунок 6.12 – Сообщение о окончании игры

ПРИЛОЖЕНИЕ В
(*обязательное*)

Диаграмма классов

ПРИЛОЖЕНИЕ Г
(*обязательное*)

Блок-схема алгоритма

ПРИЛОЖЕНИЕ Д
(*обязательное*)

Ведомость документов

ПРИЛОЖЕНИЕ Е
(*обязательное*)

Диаграмма последовательностей