Задача *Арканоид.*

В файле `arkanoid.cpp` лежит заготовка игры Arkanoid, написанная на `C++` с использованием библиотеки SFML. Ваша задача заключается в том, чтобы доработать эту игру, добавив различные виды бонусов, блоков, шариков и ракеток.

Описание классов игры

Классы программы:

- Класс `Ball` – описывает поведение одного шарика
- Класс `Paddle` – описывает поведение ракетки
- Класс `Brick` – описывает поведение блока
- Класс `BrickGrid` – описывает поведение сетки из блоков
- Класс `Bonus` – описывает поведение бонуса
- Класс `Arkanoid` – описывает поведение всей игры.

Класс `Paddle`

Класс `Paddle` описывает свойства и поведение ракетки. Основные методы этого класса:

- Конструктор `Paddle(sf::Vector2f position, sf::Vector2f size)`. Аргумент `position` задаёт центр ракетки, а `size` – её размер.
- Метод `sf::FloatRect getBorder() const` возвращает прямоугольник границ ракетки.
- Метод `void draw(sf::RenderWindow& window)` – рисует ракетку на окно `window`

Класс `Ball`

Класс `Ball` описывает свойства и поведение шарика. Основные методы этого класса:

- Конструктор `Ball(float radius, sf::Vector2f position, sf::Vector2f velocity)`
- Метод `void update(float dt)` передвигает шарик в соответствеии его скорости
- Метод `void draw(sf::RenderWindow& window)` – рисует шарик в окне `window`
- Метод `std::pair<sf::Vector2f, bool> findClosestPoint(const sf::FloatRect& rect) const` – находит ближайшую точку прямоугольника, задаваемому объектом типа `sf::FloatRect`, от центра шарика. Возвращает эту точку и булевое значение. Булевое значение показывает, пересекается ли шарик с этим прямоугольников.
- Метод `bool handleRectCollision(const sf::FloatRect& rect)` – обрабатывает упругое столкновение шарика с прямоугольником. Задаёт новые положения и скорости шарика. Возвращает `true`, если столкновение произошло и `false` иначе.
- Метод `void handleWallsCollision(sf::FloatRect boundary)` – обрабатывает упругое столкновение шарика со стенками.
- Метод `std::pair<int, int> handleBrickGridCollision(const BrickGrid& brickGrid)` – обрабатывает упругое столкновение шарика с сеткой блоков. Если произошло столкновение с блоком, то возвращает координаты блока в сетке блоков. Если столкновение не произошло, то возвращаем пару `-1, -1`.
- Метод `void handlePaddleCollision(const Paddle& paddle)` – обрабатывает столкновение с ракеткой. Столкновение не упругое. Угол отражения зависит от места на ракетке, куда стукнулся шарик. Если шарик стукнулся в левую часть ракетки, то он должен полететь влево. Если же шарик стукнулся в правую часть ракетки, то вправо.

Класс `BrickGrid`

Класс `BrickGrid` описывает свойства и поведение прямоугольной сетки блоков. Находить пересечение шарика с такой сеткой можно гораздо быстрее, чем если бы все блоки хранились поотдельности и их положение задавалось бы произвольными координатами. Благодаря регулярной сетке можно сразу определить с какими блоками может пересечься шарик, а не проверять пересечение всех шариков со всеми блоками.

Поля класса `BrickGrid`:

- `sf::FloatRect m_border` – прямоугольник, задающий область в которой находится сетка
- `int m_numBrickColumns` – размер сетки по горизонтали
- `int m_numBrickRows` – размер сетки по вертикали
- `std::vector<Brick> m_bricks` – вектор всех блоков, размер вектора `m_numBrickColumns * m_numBrickRows`.
- `sf::RectangleShape m_brickShape` – фигура SFML для отрисовки блоков.
- `int m_numActiveBricks` – текущее количество активных блоков

Методы класса `BrickGrid`:

- Конструктор `BrickGrid(sf::FloatRect borders, int numBrickColumns, int numBrickRows)`
- Метод `sf::Vector2f getBrickSizes() const` – возвращает размеры одного блока
- Метод `void deactivateBrick(std::pair<int, int> indexes)` – выключает блок с соответствующими индексами.
- Метод `void draw(sf::RenderWindow& window)` – рисует сетку блоков на окне `window`.

Класс `Bonus`

Класс `Bonus` описывает свойства бонуса. В базовой версии программы тип бонуса будет только один – бонус, который утраивает шарики. Данный класс будет являться дружественным основному классу `Arkanoid`, чтобы бонус мог менять состояние игры.

Поля класса `Bonus`:

- `sf::Vector2f m_position` – положение центра бонуса
- `float m_time` – время, прошедшее с создания бонуса в секундах

Методы класса `Bonus`:

- Конструктор `Bonus(sf::Vector2f position))`
- Метод `void update(float dt)` – перемещает бонус.
- Метод `void draw(sf::RenderWindow& window) const` – рисует бонус на окне `window`
- Метод `void activate(Arkanoid& game)` – активирует бонус, изменяет состояние игры.
- Метод `bool isColiding(const Paddle& paddle) const` – проверяет произошло ли столкновение бонуса с ракеткой.

Класс `Arkanoid`

Класс `Arkanoid` – основной класс игры, описывает поведение игры и взаимодействие всех объектов игры.

Поля класса `Arkanoid`:

- `double m_time` – время, прошедшее с начала игры в секундах
- `sf::FloatRect m_border` – прямоугольник, задающий границы игрового поля
- `std::list<Ball> m_balls` – связный список всех шариков (объектов типа `Ball`). Связный список используется так, как в процессе игры будет происходить частое добавление и удаление объектов из этого списка. В качестве альтернативы, можно было использовать `std::vector`, но удалять/добавлять шарики в конце каждого кадра.
- `BrickGrid m_brickGrid` – объект, задающий состояние сетки блоков
- `Paddle m_paddle` – объект, задающий состояние ракетки.
- `GameState m_gameState` – состояние игры. Может принимать следующие значения:
  - `GameState::stuck` – начало игры, когда шарик прикреплён к ракетке
  - `GameState::running` – процесс игры
  - `GameState::endLose` – окончание игры (поражение)
  - `GameState::endWin` – окончание игры (победа)
- `int m_numLives` – текущее количество жизней.
- `std::list<Bonus*> m_bonuses` – связный список указателей на бонусы. Указатели используются для реализации полиформизма. Так как в будущем мы хотим сделать несколько вариантов бонусов.
- `float m_bonusProbability` – вероятность выпадения бонуса после разрушения одного блока.
- `Ball m_initialBall` – объект шарика, используемый для рисования шарика на ракетке в состоянии `GameState::stuc` Также этот объект используется для рисования шариков в полоске количества жизней.
- `sf::Text m_endText` – текст, используемый для отображения сообщения о победе/поражении.

Методы класса `Arkanoid`:

- Конструктор `Arkanoid(sf::FloatRect border, sf::Font& font)`
- `void addBall(const Ball& ball)` – добавляет новый шарик в игру.
- `void update(const sf::RenderWindow& window, float dt)` – функция, которая вызывается каждый кадр. Она перемещает все объекты, обрабатывает столкновение всех объектов и меняет состояние игры.
- `void draw(sf::RenderWindow& window)` – рисует все объекты игры на окно `window`.
- `void onMousePressed(sf::Event& event)` – обрабатывает состояние нажатия мыши.

Задача 1: Бонусы

Используйте наследование и полиморфизм и добавьте в игру следующие бонусы:

- Бонус, который увеличивает количество шариков в 3 раза (этот бонус уже реализован)
- Бонус, который увеличивает размер ракетки
- Бонус, который уменьшает размер ракетки
- Бонус, который временно замедляет движение шариков
- Бонус, который временно добавляет "пол" под ракеткой, от которого могут отскакивать шарики. В этом случае шарики временно вообще не могут упасть вниз.
- Бонус, который временно делает все шарики красными. Красные шарики пробивают блоки насквозь, то есть не отскакивают от них, а просто уничтожают их и проходят дальше без изменения скорости.
- Бонус, который временно делает ракетку зелёной. Шарики прилепают к зелёной ракетке и остаются на ней пока пользователь не нажмёт левую кнопку мыши.

Для этого сделайте класс `Bonus` абстрактным и отнаследуйте от него классы конкретных бонусов. Например класс `EnlargePaddleBonus` будет описывать класс бонуса, увеличивающего размер ракетки. Этот класс должен наследоваться от абстрактного класса `Bonus`. Указатели из списка `m_bonuses` класса `Arkanoid` будут указывать на разные типы бонусов. Вам придётся немного изменить и другие классы, чтобы некоторые бонусы работали.

Задача 2: Блоки

Используйте наследование и полиморфизм и добавьте в игру разные виды блоков:

- Обычный блок, разрушается с одного раза (этот блок уже реализован)
- Блок, который разрушается только с трёх ударов. За исключением когда по нему ударил красный шарик, тогда блок разрушается сразу.
- Неразрушаемый блок. Не разрушается от шариков. Для победы в игре уничтожать эти блоки не нужно.
- Взрывающийся блок. Блок, который при столкновении с шариком активируется и через некоторое короткое время взрывается и уничтожает соседние блоки. Если один из соседний блоков тоже взрывающийся, то он должен также взорваться. Создать также простую анимацию взрыва (например, можно нарисовать круг на короткое время в месте взрыва).

Задача 3: Снаряд (`Bullet`)

Добавьте в игру новый бонус, который будет временно давать возможность ракетке стрелять. При нажатии левой кнопки мыши ракетка должна выстреливать вверх по 2 снаряда с левого и правого конца ракетки. При столкновении с блоками эти снаряды должны уничтожать блоки. Для этого создайте новый класс `Bullet` который будет описывать поведение снаряда.

Задача 4: Уровень

Создайте новый класс под названием Уровень (`Level`). В этом классе должно храниться расположение и типы всех блоков. Класс должен уметь загружать уровень из файла. Используйте этот класс, чтобы создать в игре несколько разных уровней.