```
  private final int _cols = 10;

  public BrickPile(PlayField pf, Image img1, Image img2) {
    _pf = pf;
    int startx = 80;
    int x = startx, y = 10;

    for (int r = 0; r < _rows; r++) {
      for (int c = 0; c < _cols; c++) {
        Rectangle pos =
          new Rectangle(x,y,img1.getWidth(null),img1.getHeight(null));
        // В зависимости от номера кирпича добавим на игровое поле
        // либо простой кирпич, либо крепкий.
        if (((r+1) * (c+1)) % (_rows * _cols * 0.2) == 0 ) {
          pf.addSprite(new HardBrick(_pf, this, pos, img1, img2));
        } else {
          pf.addSprite(new Brick(_pf, this, img1, pos));
        }
        x += img1.getWidth(null);
      }
      y += img1.getHeight(null) + 2;
      x = startx;
    }
  }
  ...
}
```

Литература и гиперссылки

[1] Морган М. *Java 2. Руководство разработчика* // М.: «Вильямс», 2000.

[2] Фаулер М., Скотт К. *UML. Основы* // СПб: Символ-Плюс, 2002.

[3] Макгрегор Д., Сайкс Д. *Тестирование объектно-ориентированного программного обеспечения* // К.: ООО «ТИД «ДС», 2002.

[4] Орлов С. *Технология разработки программного обеспечения* // СПб.: Питер, 2002.

[5] Роганов Е. *Основы информатики и программирования* // М.: МГИУ, 2001.

[6] Эферган М. *Java* // СПб.: Питер, 1998.

[7] `http://www.sun.com` — Официальный сайт разработчиков языка Java.

[8] `http://docs.rinet.ru:8083/J21/ch24.htm` — Статья, посвящённая спрайтам и анимации.

§ 1. Игра «Кирпичики»

В этом разделе будет описана интерактивная компьютерная игра «Кирпичики»(Bricles). Она очень похожа на игру Breakout, одну из первых коммерческих видеоигр, получивших распространение среди пользователей компьютеров Apple II.

1. **Базовые компоненты игры «Кирпичики».** «Кирпичики» — это аркадная игра (игра для компьютерных автоматов). Игровое поле представляет собой прямоугольник, ограниченный двумя стенами, полом и потолком. На поле находится стопка кирпичей, называемая «кучей». Цель играющего заключается в том, чтобы выбить все кирпичи из кучи, попадая в каждый из них шайбой. Шайба приводится в движение при помощи лопатки, управляемой игроком. При движении она отражается от стен, потолка, кирпичей (которые при этом разбиваются) и лопатки. Падая на пол, шайба выходит из игры.

Игрок получает в свое распоряжение три шайбы, первая из которых в начале игры помещается в центр игрового поля и начинает двигаться вниз. Игрок, управляя лопаткой с помощью клавиатуры, должен перемещать её так, чтобы шайба отражалась от неё, а не падала на пол. Когда шайба ударяется об лопатку, она устремляется вверх. В случае падения шайбы на пол она выходит из игры и заменяется на одну из оставшихся шайб. Игра завершается проигрышем, если начальный запас шайб будет израсходован до полного разрушения всех кирпичей.

2. **Физические основы игры.** Во время своего движения по игровому полю шайба соприкасается с его различными компонентами. При этом они взаимодействуют между собой описанным ниже образом.

- **Потолок и стены.** Шайба отражается от потолка и стен в соответствии с законами физики без учета сил трения и тяжести, в результате чего угол падения равен углу отражения (рис. 1).
- **Пол.** Пол поглощает шайбу. Шайба, упавшая на пол, от него не отражается, а выходит из игры.
- **Кирпичи.** Шайба отражается от кирпича таким образом, что угол её падения равен углу отражения. При столкновении шайбы и кирпича последний разрушается. Обратим внимание на то, что шайба может ударять кирпич как сверху, так и снизу. Кирпич обладает достаточной толщиной, чтобы шайба могла попасть в него сбоку[1] (рис. 2).

[1] Реализация отскока шайбы от боковых стенок кирпича является одним из заданий курсовой работы.

кирпича и шайбы. Представим эти изменения при помощи диаграммы классов (рис. 12).

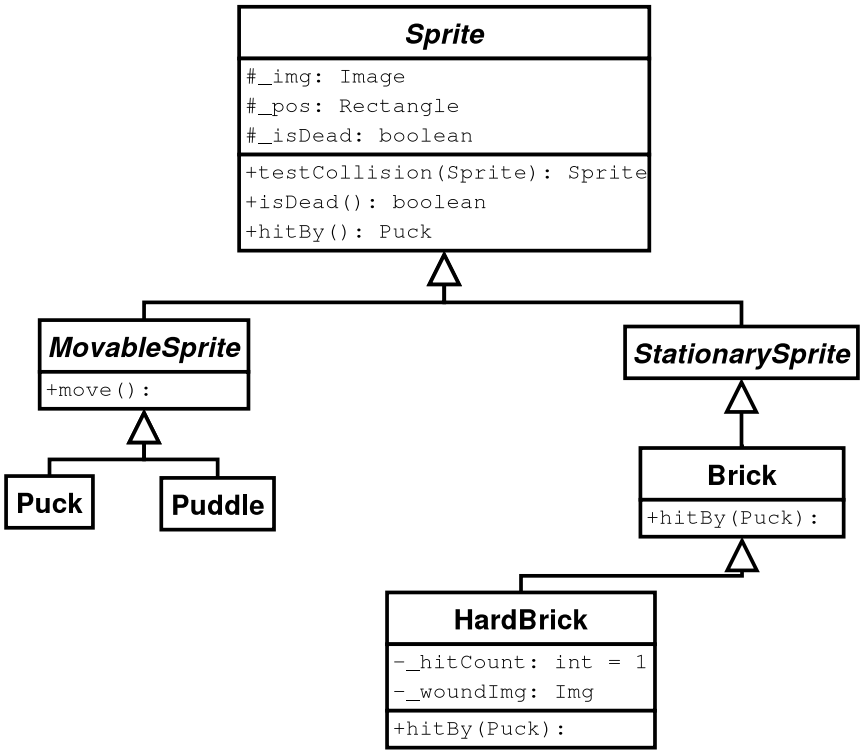

Рис. 12. Добавление HardBrick в иерархию спрайтов

Основные изменения в программном коде сводятся к модификации метода `hitBy`.

```
import java.awt.Image;
import java.awt.Rectangle;

class HardBrick extends Brick {
  /*
   @_hitCount - количество допустимых ударов шайбой
                кирпич разрушается, как только атрибут
                будет равен нулю
   @_woundImg - изображение повреждённого кирпича
```


неё, двигаясь слева, либо правую часть, если шайба двигается справа. *Дальняя треть* определяется аналогично, а *средней* является оставшаяся часть лопатки. Отражение шайбы будем определять по следующим правилам[2] (см. рис. 3):

- если шайба попадает в ближнюю треть лопатки, то она отражается точно в противоположном направлении;
- если шайба попадает в среднюю треть, то угол отражения немного больше угла падения, однако новое направление движения не может быть вертикальным;
- если шайба попадает в дальнюю треть лопатки, то угол отражения несколько меньше угла падения, но направление движения шайбы не может стать горизонтальным.

- **Шайба.** В начале игроку выделяется несколько шайб, однако в игре постоянно участвует только одна шайба. Как только она падает на пол, в игру включается следующая шайба.

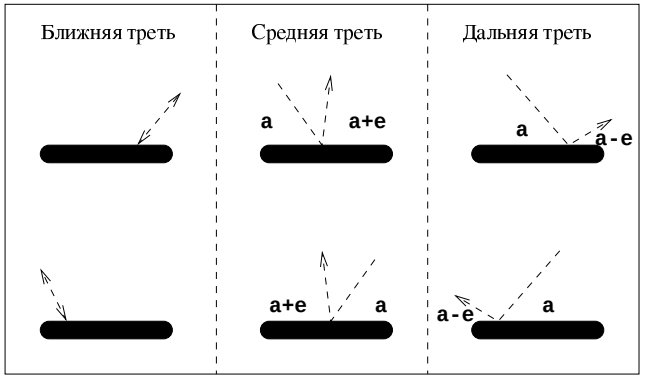

Рис. 3. Взаимодействие шайбы с лопаткой

3. **Действия игрока.** Игра начинается после нажатия игроком клавиши S (start). Игрок может выйти из игры в любой момент до того, как он выиграет или проиграет, он может приостановить игру (объявить «паузу») и возобновить её после этого. Если игрок выигрывает, то на экран выводится поздравление, в противном случае программа выражает ему своё сочувствие.

[2]Данные правила не реализованы полностью в эталонном проекте — это является ещё одним из заданий на модификацию.

Задание 3. Добавить новый вид кирпичей. Кирпич (WallBrick) нельзя разбить шайбой. Игра заканчивается, когда разбиты все кирпичи, кроме таких.

Задание 4. Добавить новый вид кирпичей. При попадании в «липучий» кирпич (StickyBrick) шайба прилипает к нему на две единицы времени, а затем вертикально падает. Данный кирпич нельзя разбить шайбой. Игра заканчивается, когда разбиты все кирпичи, кроме «липучих».

Задание 5. Добавить новый вид кирпичей. Бронированный кирпич (ArmorBrick) можно разбить только в том случае, когда шайба попадёт в него сверху (со стороны потолка).

Задание 6. Добавить новый вид кирпичей. При попадании шайбы в кирпич-ловушку (TrapBrick) снизу шайба теряется. Кирпич уничтожается при ударе шайбы с любой другой стороны.

Задание 7. Добавить редактор уровней. Информация о положении кирпичей на игровом поле должна содержаться в обычном текстовом файле в следующем формате: `B` означает наличие кирпича, а пробел — отсутствие. Файл конфигурации загружается при помощи меню в верхней панели.

Задание 8. Добавить «мультфильмы», т.е. возможность воспроизведения игры (передвижения шайбы и лопатки, разбивание кирпичей) после её окончания.

Задание 9. Добавить таблицу очков. В случае выигрыша программа запрашивает у игрока имя и записывает его в файл `score.txt` вместе с числом набранных им очков, которое зависит от количества разбитых кирпичей и затраченного времени. Таблицу очков нужно просматривать с помощью меню.

Задание 10. Добавить свойство прилипания шайбы к лопатке. Игрок нажимает клавишу «пробел», после чего при столкновении шайбы с лопаткой шайба не отскакивает, как обычно, а остаётся на лопатке. Игрок может прицелиться, передвигая лопатку, и затем, выбрав цель, «выстрелить» шайбой, повторно нажав клавишу «пробел». Этой возможностью можно воспользоваться только один раз.

Задание 11. Изменить свойства стен таким образом, чтобы при отскоке шайба вылетала из противоположной стены, сохраняя при этом направление и скорость (рис. 11).

Задание 12. Добавить «уровни». Каждый раз когда игрок выигрывает, программа создаёт новый уровень, случайным образом расставляя кирпичи в некотором ограниченном прямоугольнике игрового пространства.

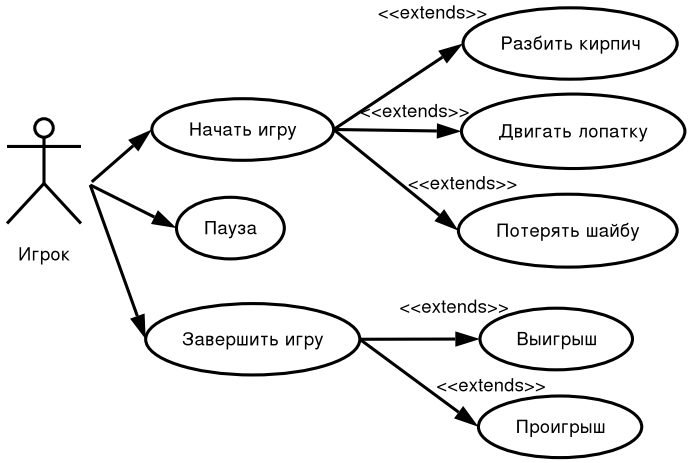

Рис. 4. Случаи использования игры «Кирпичики»

лопатку» порождает следующие сценарии перемещения лопатки влево перед её столкновением с шайбой, движущейся слева направо:

- перемещение, обеспечивающее столкновение в средней трети лопатки;
- перемещение, обеспечивающее столкновение в дальней трети лопатки;
- перемещение, обеспечивающее столкновение в ближней трети лопатки.

Сценарии применяются к объектам, в которых значения атрибутов конкретизированы. Это отличает их от случаев использования, которые ориентированы на работу с объектами без конкретных значений атрибутов. Случаи использования обычно формулируют на естественном языке, а затем представляют в графической форме в виде диаграммы (рис. 4).

2. **Объекты и классы.** Описав требования при помощи диаграмм использования, приступим к созданию модели классов игры, часто называемой *диаграммой классов*. Сначала напомним базовые концепции объектно-ориентированного программирования и способы графического представления различных компонент объектного мира при помощи языка UML.

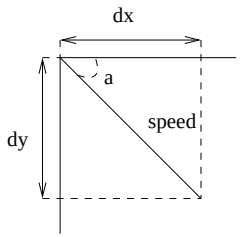

Рис. 10. Задание скорости

```
/*
 @_dx - составляющая скорости по оси X
 @_dy - составляющая скорости по оси Y
 @_speed - скорость спрайта
 */

private double _dx, _dy;
private int _speed;

public Velocity(int D, int S) {
  _speed = S;
  setDirection(D);
}

public int getSpeed() {
  return _speed;
}

public int getSpeedX() {
  return (int) _dx;
}

public int getSpeedY() {
  return (int) _dy;
}

public void setDirection(int d) {
  _dx = Math.cos(Math.toRadians(d)) * (double) _speed;
  _dy = Math.sin(Math.toRadians(d)) * (double) _speed;
}

public int getDirection() {
```


- защищённую (protected), используемую лишь объектами данного класса и выведенными из него (знак «#»);
- закрытую (private), доступную только самому классу (знак «−»).

Интерфейс класса `PuckSupply` (склад шайб) изображён на рис. 5. Этот класс представляет собой набор шайб, выдаваемых игроку в начале игры. Когда шайба выходит из игры (падает на пол), программа заменяет её другой, если их запас ещё не исчерпан.

А вот как может выглядеть реализация данного класса на языке Java.

```
class PuckSupply {
  /*
   @_puck - массив шайб
   @_count - кол-во оставшихся у игрока шайб
   */
  private Puck _pucks[];
  private int _count;

  public PuckSupply(int N, PlayField pf, Image pic) {
    _pucks = new Puck[N];
    for (int i = 0; i < N; i++)
      _pucks[i] = new Puck(pf, this, pic);
    _count = N;
  }

  public int size() {
    return _count;
  }
  /* Взять следующую шайбу из хранилища */
  public Puck get() {
    return _count > 0 ? _pucks[--_count] : null;
  }
}
```

3. **Отношения между классами.** Классы, используемые в том или ином проекте, не существуют изолированно друг от друга. Между ними всегда имеются определённые отношения, среди которых можно выделить *ассоциацию*, *наследование*, *агрегацию* и *зависимость*.

Ассоциации обеспечивают взаимодействие объектов, принадлежащих разным классам. Они являются «клеем», соединяющим воедино все элементы программной системы. Примером ассоциации является связь класса `PuckSupply` и `Puck` (шайба). `PuckSupply` играет роль хранилища, в котором находятся шайбы. Данная ассоциация (рис. 6) предполагает двустороннюю связь. Для экземпляра класса, реализующего склад шайб,

```
   @_pf  - игровое поле
   @_pos - позиция и размеры спрайта
   @_isDead - состояние - "мертв", "жив"
   * Мертвый спрайт удаляется с игрового поля.
   */

  protected Image _img;
  protected PlayField _pf;
  protected Rectangle _pos;
  protected boolean _isDead;
  ...
  public void draw(Graphics g) {
    g.drawImage(_img, _pos.x, _pos.y, _pf);
  }
  /* Проверка на наличие коллизии. */
  public boolean testCollision(Sprite s) {
    if (s != this)
      return _pos.intersects(s.getBounds());
    return false;
  }
  /* Взять ограничивающий прямоугольник спрайта. */
  public Rectangle getBounds() {
    return _pos;
  }
  /* Мертв или жив? */
  public boolean isDead() {
    return _isDead;
  }
  ...
}
```

`MovableSprite` расширяет базовый класс `Sprite`, добавляя возможность движения при помощи метода `move`, который контролирует также столкновения как с границами игрового поля, так и с другими спрайтами. В случаях столкновения с другими спрайтами вызывается метод `collideInto`, а столкновения с границами игрового поля обрабатываются отдельно.

```
class Puck extends MovableSprite {
  ...
  public void move() {
    if (!_isMoving)
      return;
    Rectangle b = _pf.getBoundary();
```


`MovableSprite` (подвижный спрайт) — это спрайт, способный изменять свое местоположение. С таким спрайтом ассоциирована скорость движения в текущий момент времени. Скорость (`Velocity`) представлена направлением движения и расстоянием, преодолеваемым за единицу времени. В нашем проекте подвижными спрайтами являются шайба (`Puck`) и лопатка (`Puddle`).

`StationarySprite` (неподвижный спрайт) не меняет своего положения на игровом поле. Единственным примером такого спрайта в проекте является кирпич (`Brick`). При изображении иерархии классов принято отношение наследования указывать стрелкой с незакрашенным наконечником, направленной от производного класса к базовому (рис. 7).

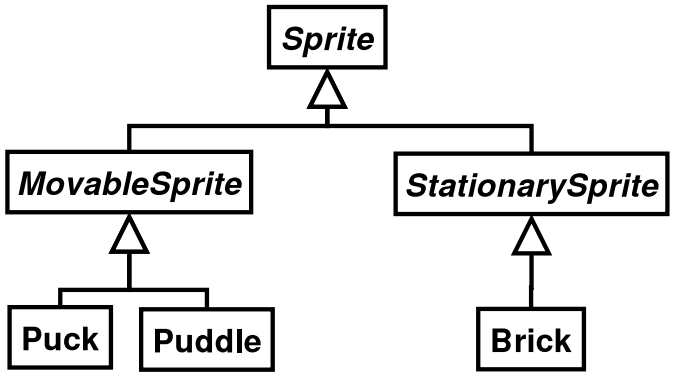

Рис. 7. Иерархия объектов игрового поля

Агрегация — описывает отношение «является частью». Например, можно сказать, что двигатель и колёса являются частью автомобиля. Кроме агрегации в языке UML определено более сильное понятие, называемое *композицией*. При композиции объект-часть может принадлежать только одному целому и, как правило, жизненный цикл части совпадает с жизненным циклом целого, т. е. части живут и умирают вместе с целым. Агрегацию изображают в виде стрелки с наконечником в форме ромба.

Зависимость — это отношение, которое показывает, что изменения в одном классе могут повлиять на другой. Графически зависимость изображается пунктирной стрелкой, идущей от зависимого класса к тому, от которого он зависит. Наиболее часто зависимости показывают, что один класс использует другой класс в качестве аргумента какого-то из своих методов. Например, в рамках нашего проекта подвижный спрайт может