IMD242_202221498_지예린_mid project intention

우주선 격파 게임 레퍼런스 (코드 참고 X)

https://blog.naver.com/kil4776/220920269083

코드가 너무 길어져서 클래스는 새 탭으로 작업했습니다.

메인 - SpaceShooter_main.pde

클래스 - player.pde

star.pde

enermy.pde

explosion.pde

Missile.pde

챗Gpt 활용 코드 레드 컬러로 표시

- **주요 ArrayList 활용**

1. 미사일 관리 (ArrayList<Missile> missiles;)

   플레이어가 발사하는 미사일을 새로 업데이트 및 화면에서 벗어나거나 적과 충돌하는 미사일을 제거하는 등의 관리.

   Missile Count : 화면 상에 남아있는 미사일 수를 통해 ArrayList의 현재 크기가 화면 좌측 상단에 표시된다. (* AllayList 관리를 위해 화면밖으로 나가면 사라지는 미사일 수가 삭제되어 미사일 발사 횟수가 아닌 화면 상에 남아있는 미사일 수로 측정됨.)

2. 적 우주선 관리 (ArrayList<Enemy> enemies;)

   적 우주선 객체를 새로 업데이트하며 화면 밖으로 나가거나 미사일에 맞으면 제거하는 등의 관리.

   Enemy Count : 화면 상에 남아있는 적 우주선 수를 보여준다. 하나가 사라지면 바로 하나가 다시 생성되는 로직으로 8개를 유지한다.

3. 폭발 효과 관리 (ArrayList<Explosion> explosions;)

   미사일과 적 우주선이 충돌했을 때 발생하는 폭발 효과를 저장하고, 일정 시간이 지나면 제거하는 것을 관리.

   Explosion Count : 현재 발동중인 폭발 효과 수를 표시한다. 폭발 효과가 일정 시간이 지나 끝나면 어레이 리스트에서 제거된다.

- **주요 Class 활용**

  1. 우주 배경 클래스 (Star)
  2. 플레이어 우주선 클래스 (Player)
  3. 적 우주선 클래스 (Enemy)
  4. 미사일 클래스 (Missile)
  5. 미사일과 적 우주선 충돌 시 폭발 클래스 (Explosion)

- **조건 예문**

  1. 우주선이 움직이는 듯한 느낌을 주기 위해 화면에 별이 화면 위에서부터 랜덤한 크기와 속도, 위치(x)로 떨어지도록 한다.
  2. 화면에는 200개의 별이 생성되고 별이 아랫변을 뚫고 사라질 때 마다 초기화되어 새로운 별이 생성되도록 한다.
  3. 플레이어 우주선은 y좌표 하단에 고정되며 좌, 우를 넘지 못한다. (화면 경계 넘지않도록)
  4. 플레이어 우주선은 마우스 x 좌표를 따라다닌다.

5. 미사일은 화면을 벗어나면 사라지도록 한다.
6. 마우스를 클릭할 때 마다 미사일이 나오도록 한다.(미사일 개수와 크기는 랜덤)
7. 미사일은 플레이어 우주선에서 나오도록 한다.
8. ArrayList를 통해 화면 상의 미사일 수를 기록 및 관리한다.
9. 미사일의 개수는 1~3개로 랜덤하며 각 미사일 간의 거리는 30으로 일정하게 한다.
10. 적 우주선의 사이즈는 랜덤하게 한다.
11. 초반에 바로 생성되는 적 우주선을 화면 상단 너머에 20개정도 만들어주며 상단 너머에서 각각 생성되는 높이를 랜덤하게 하여 다른 위치에 보이도록 한다.
12. 적 우주선은 미사일과 충돌하거나 화면을 벗어나면 다시 생성된다.
13. 플레이어의 미사일과 적 우주선이 충돌 시 둘다 제거된다.
14. 폭발 효과는 점점 커지는 모션 효과를 사용한다.
15. 폭발 효과는 미사일과 적 우주선이 충돌된 지점에서 생성된다.
16. 폭발 효과는 효과가 끝나면 Arraylist에서 제거된다.
17. 화면 좌측 상단에 각 ArrayList 크기를 기재한다.

- **제작순서**

    1. 배경 제작
    2. 플레이어 우주선 제작
    3. (플레이어 우주선에서 나오는) 미사일 제작
    4. 적 우주선 제작
    5. 플레이어의 미사일과 적 우주선 충돌 시 제거되는 코드 제작
    6. 충돌 시 생성되는 폭발 효과 제작
    7. 화면 좌측 상단에 점수와 ArrayList 크기 기재

---

1. 배경 제작

- 우주선이 움직이는 듯한 느낌을 주기 위해 화면에 별이 화면 위에서부터

랜덤한 크기와 속도, 위치(x)로 떨어지도록 한다.

- 화면에는 200개의 별이 생성되고 별이 아랫변을 뚫고 사라질 때 마다 초기화되어 새로운 별이 생성되도록 한다.

배경을 만들기 위해 star 클래스를 만들어줌. (클래스는 새 탭에 따로 제작)

```
class Star {
  float x, y, speed, size;

  Star() {
    x = random(width);
    y = random(-height); // -를 붙여 화면 위에서 시작하도록 함.
    speed = random(1, 3);    // 떨어지는 속도 랜덤하게
    size = random(1, 3);     // 별의 크기 랜덤하게
  }

//별의 움직임. 화면 아래로 떨어지면 초기화되어 다시 화면위에서 나타나게.
```

```
  void move() {
    y += speed; // 별이 아래로 떨어지는 동작 실행.
    if (y > height) {
      y = random(-50, 0); // 화면 아래로 떨어지면 화면 위로 돌아가기
      x = random(width);
      speed = random(1, 3);
      size = random(1, 3);
    }
  }

  void display() {
    noStroke();
    fill(255, 255, 200, 200); //별에 약간 노란빛을 더해줌
    ellipse(x, y, size, size);
  }
}
```

배경 메인 코드 제작

```
Star[] stars = new Star[200]; // 200개의 별이 생성되어 관리되도록 함.



// 셋업에서 위에서 언급한 200개의 별들이 각각 초기화되어 실행되도록 함.

void setup() {

   for (int i = 0; i < stars.length; i++) {

      stars[i] = new Star();

   }

}



// star 클래스에 만든 move, display 불러와서 별을 계속 그리도록 함.

void draw() {

   background(5);

   for (int i = 0; i < stars.length; i++) {

      stars[i].move();

      stars[i].display();

   }

}
```

---

2. 플레이어 우주선 제작.

- 플레이어 우주선은 y좌표 하단에 고정되며 좌, 우를 넘지 못한다. (화면 경계 넘지않도록)
- 플레이어 우주선은 마우스 x 좌표를 따라다닌다.

플레이어 우주선만들기 위해 player 클래스 만들어줌. (새 탭)

```
class Player {

  float x, y; // 플레이어의 위치. 업데이트에서 mouseX,Y로 설정해줄 예정.

  float size = 40; // 플레이어 우주선 크기가 40이 되도록 함.



  Player() {
```

```
  x = width / 2; // 처음 켜면 중간에 놓치도록 함.
  y = height - 100; // 화면 최하단에서 -100지점에 놓치도록 함.
}

void update() {
  x = mouseX;       // 마우스의 X 좌표를 따라다니도록

  // 화면 경계를 벗어나지 않도록 제한함. 챗gpt 활용
  if (x < size * 0.9) {
    x = size * 0.9;
  } else if (x > width - size * 0.9) {
    x = width - size * 0.9;
  }
}

void display() {
  // 플레이어 우주선 본체 그리기.
  fill(100, 150, 255);
```

```
      stroke(255);
      strokeWeight(2);
      triangle(
        x, y - size / 3,
        x - size / 2, y + size,
        x + size / 2, y + size
        );

      // 엔진 불꽃 위치 설정함. 엔진 불꽃은 따로 그려줌.
      displayEngine(x -15, y + 45);
      displayEngine(x + 15, y + 45);
   }
}

// 플레이어 우주선 엔진 그리기.
void displayEngine(float xE, float yE) {
   // r값 높게 랜덤, g값 낮게 랜덤, b값 0으로 해서 주황색 채도가 랜덤하게 나오도록 함.
```

```
  fill(random(200, 255), random(100, 200), 0);
  noStroke();
  beginShape();
  vertex(xE - 5, yE);
  vertex(xE + 5, yE);
  vertex(xE, yE + random(10, 20)); // 엔진 불꽃 높이 랜덤하게 해서 나아가는 것처럼.
  endShape(CLOSE);
}
```

플레이어 우주선 메인 코드

```
Player player;   //최상단에 플레이어 우주선 클래스 불러옴.


//플레이어 실행되도록 셋업에 추가해줌.
void setup() {
player = new Player();
```

```
}



//플레이어 클래스에서 만든 플레이어 우주선 그리도록 드로우에 추가해줌.

void draw() {

player.update();

player.display();

}
```

---

3. 미사일 제작

- 미사일은 화면을 벗어나면 사라지도록 한다.
- 마우스를 클릭할 때 마다 미사일이 나오도록 한다.(미사일 개수와 크기는 랜덤)
- 미사일은 플레이어 우주선에서 나오도록 한다.
- ArrayList를 통해 화면 상의 미사일 수를 기록 및 관리한다.

- 미사일의 개수는 1~3개로 랜덤하며 각 미사일 간의 거리는 30으로 일정하게 한다.

미사일 만들기 위해 missile 클래스 제작. (새 탭)

```
class Missile {

  float x, y; // 미사일 위치 지정을 위함.

  float size; // 미사일의 크기를 랜덤하게 주기 위함.

  float speed; // 미사일 속도를 위함. (일정하게 줄 것)



  Missile(float x, float y, float size) {

    this.x = x;

    this.y = y;
```

```
    this.size = size;
    this.speed = 5; //미사일 속도는 일정하게.
  }

  void update() {
    y -= speed; //화면 위쪽으로 이동하도록 해줌.
  }

  void display() {
    fill(random(200, 255), random(100, 200), 0); //엔진과 동일한 색상 랜덤 활용함.
    noStroke();
    ellipse(x, y, size, size * 2.5); //타원형 미사일으로 제작함.
  }

//화면을 벗어나면 true 반환하도록 해서 어레이리스트통해 제거되도록 챗gpt 활용함.
  boolean offScreen() {
    return y + size < 0; //미사일이 화면 밖으로 나갔는지.
```

```
  }

}
```

미사일 메인 코드.

```
ArrayList<Missile> missiles; // 미사일 생성 제거 관리를 위해 미사일은 어레이리스트로 불러옴.



void setup() {

missiles = new ArrayList<Missile>();

}



// 마우스 클릭으로 미사일 발사되도록 함.

일정한 간격을 유지하며 미사일을 생성하기 위해 챗gpt 활용하였음.

void mousePressed() {

// 마우스를 클릭하면 1-3 값의 랜덤한 수의 미사일을 생성하도록 함.
```

```
  int missileCount = int(random(1, 4)); // 1~3개의 미사일 생성함.
  float spacing = 30; // 미사일 간격
  float totalWidth = (missileCount - 1) * spacing; //30간격 주기 위해 전체 폭에서 미사일 수 나눠줌.
  float startX = player.x - totalWidth / 2; // 첫 번째 미사일의 시작 위치를 플레이어 우주선 가운데로 지정함.

  for (int i = 0; i < missileCount; i++) {
    float missileX = startX + i * spacing; // 각 미사일의 x 좌표를 구해줌.
    float missileSize = random(3, 7); // 미사일 크기 살짝 랜덤하게 해줌.
    missiles.add(new Missile(missileX, player.y, missileSize));
  }
}

  //미사일 클래스에서 만든거 그려줌.
void draw() {
  for (int idx = 0; idx < missiles.size(); idx++) {
    Missile aMissile = missiles.get(idx);
```

```
      aMissile.update(); //미사일 이동하는 것 불러옴.
      aMissile.display(); //미사일 그린 것 불러옴.

      //미사일이 화면 밖으로 나가서 true 반환되면 제거되는 것을 구현하기 위해 챗gpt 활용함.
      if (aMissile.offScreen()) {
         missiles.remove(idx);
      }
   }
}
```

---

4. 적 우주선 제작

- 적 우주선의 사이즈는 랜덤하게 한다.
- 초반에 바로 생성되는 적 우주선을 화면 상단 너머에 20개정도 만들어주며 상단 너머에서 각각 생성되는 높이를 랜덤하게 하여 다른 위치에 보이

도록 한다.

- 적 우주선은 미사일과 충돌하거나 화면을 벗어나면 다시 생성된다.

적 우주선을 만들고 챗gpt에게 x,y,size로 재정의해달라하여 활용함.

```
void display() {
//본체1
fill(50, 255, 50);
noStroke();
ellipse(300, 300, 200, 100);
//본체2
fill(100, 255, 100);
ellipse(300, 250, 160, 60);
//창문
fill(255); ellipse(240, 270, 20, 20);
ellipse(300, 270, 20, 20);
ellipse(360, 270, 20, 20);
//다리
```

```
fill(255, 0, 0, 150);

ellipse(240, 330, 30, 30);

ellipse(360, 330, 30, 30); }



->



// 본체1 (큰 타원)

    fill(random(0,70), 180, random(0,70));

    stroke(0);

    ellipse(x, y, size * 2, size);



    // 본체2 (작은 타원)

    fill(120, 255, 150, 98);

    stroke(255, 50);

    ellipse(x, y - size / 2, size * 1.6, size * 0.6);



    // 창문 (원 3개)

    fill(255);
```

```
    float windowSize = size * 0.1;
    ellipse(x - size * 0.3, y - size / 3, windowSize, windowSize);
    ellipse(x, y - size / 3, windowSize, windowSize);
    ellipse(x + size * 0.3, y - size / 3, windowSize, windowSize);

    // 다리 (원 2개)
    fill(255, 0, 0, 150);
    float legSize = size * 0.2;
    ellipse(x - size * 0.3, y + size / 3, legSize, legSize);
    ellipse(x + size * 0.3, y + size / 3, legSize, legSize);
```

적 우주선 만들기 위해 enemy 클래스 제작. (새 탭)

```
class Enemy {
  float x, y;
  float speed = 3; //적 우주선 속도 일정하게 함.
```

```
  float size = random(20, 40); //적 우주선 크기 랜덤하게 함.



  Enemy(float x, float y) {

    this.x = x;

    this.y = y;

  }



  void update() {

    y += speed; //아래로 이동하도록 하는 코드.

  }



//적 우주선 챗지피티에게 x,y,size로 재정의해달라고 했던 코드.

  void display() {

    // 본체1 (큰 타원)

    fill(random(0,70), 180, random(0,70));

    stroke(0);

    ellipse(x, y, size * 2, size);
```

```
    // 본체2 (작은 타원)
    fill(120, 255, 150, 98);
    stroke(255, 50);
    ellipse(x, y - size / 2, size * 1.6, size * 0.6);

    // 창문 (원 3개)
    fill(255);
    float windowSize = size * 0.1;
    ellipse(x - size * 0.3, y - size / 3, windowSize, windowSize);
    ellipse(x, y - size / 3, windowSize, windowSize);
    ellipse(x + size * 0.3, y - size / 3, windowSize, windowSize);

    // 다리 (원 2개)
    fill(255, 0, 0, 150);
    float legSize = size * 0.2;
    ellipse(x - size * 0.3, y + size / 3, legSize, legSize);
    ellipse(x + size * 0.3, y + size / 3, legSize, legSize);
}
```

적 우주선 메인 코드.

```
ArrayList<Enemy> enemies; //적 우주선 관리하는 ArrayList 생성해줌.



void setup() {

enemies = new ArrayList<Enemy>();



//초반에 바로 생성되는 적 우주선을 화면 상단 너머에 5개정도 만들어줌.

// spawnEnemy부분과 함께 챗gpt를 활용함.

  for (int i = 0; i < 5; i++) {

    spawnEnemy();

  }



  // 랜덤 위치에 새로운 적 우주선 생성

void spawnEnemy() {

  enemies.add(new Enemy(random(50, width - 50), random(-200, -50)));
```

```
}



// 적 우주선 클래스 만든거 그려줌.
void draw() {
   for (int i = enemies.size() - 1; i >= 0; i--) {
      Enemy aEnemy = enemies.get(i);
      e.update();
      e.display();

//적 우주선이 화면을 벗어났을 때 그 적을 제거하고, 새로운 적 우주선을
생성하는 로직을 챗gpt활용.
      if (aEnemy.y > height) {
         enemies.remove(i); //화면을 벗어난 적 우주선 제거하는 코드.
         spawnEnemy(); //새로운 적 우주선 추가해주는 코드.
//continue를 사용해 현재 적 우주선에 대한 처리를 마치고 다음 적 우주선을 처
리함.
         continue;
```

```
}
```

---

5. 플레이어의 미사일과 적 우주선 충돌 시 제거되는 코드 제작

- 플레이어의 미사일과 적 우주선이 충돌 시 둘다 제거된다.
- 적 우주선은 미사일과 충돌하거나 화면을 벗어나면 다시 생성된다.

적 우주선과 미사일의 충돌을 체크하는 코드를 챗gpt 활용하여 제작함.

Enemy 클래스에서의 코드.

```
//적 우주선과 미사일의 충돌을 체크하는 것을 챗gpt에게 물어봄.
  //미사일과 적 우주선 사이의 거리를 계산하여 충돌 여부를 판단함.
  boolean isHit(Missile m) {
    float d = dist(x, y, m.x, m.y);
    return d < size / 2;
  }
}
```

메인 코드1. 해당 코드는 적 우주선이 미사일과 충돌하지않고 화면을 벗어났을 때의 경우

```
//적 우주선이 화면을 벗어났을 때 그 적을 제거하고, 새로운 적 우주선을 생성하는 로직을 챗gpt활용.
void draw() {
    if (aEnemy.y > height) {
      enemies.remove(i); //화면을 벗어난 적 우주선 제거하는 코드.
      spawnEnemy(); //새로운 적 우주선 추가해주는 코드.
      continue; //continue를 사용해 현재 적 우주선에 대한 처리를 마치고 다음
```

```
적 우주선을 처리함.
    }
}
```

메인 코드2. 해당 코드는 적 우주선이 미사일과 충돌하여 사라진 경우.

```
// 적 우주선과 미사일 충돌하면 우주선이 제거되고 새로운 우주선 생성해주는 코드.
    // 충돌 시 점수를 증가시킴.
void draw() {
    for (int j = missiles.size() - 1; j >= 0; j--) {
        Missile m = missiles.get(j);
        //충돌 발생 시
        if (aEnemy.isHit(m)) {
            enemies.remove(i);   //충돌된 적 우주선 제거 코드.
            missiles.remove(j); //충돌된 미사일 제거 코드.
            score += 100;          //충돌할 때 마다 100점 씩 점수를 증가시킴.
            spawnEnemy();            //하나 사라지면 새로운 적 우주선을 추가해주는
```

```
코드.
            break;
        }
}
```

---

6. 충돌 시 생성되는 폭발 효과 제작

- 폭발 효과는 점점 커지는 모션 효과를 사용한다.
- 폭발 효과는 미사일과 적 우주선이 충돌된 지점에서 생성된다.
- 폭발 효과는 효과가 끝나면 Arraylist에서 제거된다.

플레이어 미사일과 적 우주선 충돌 시 폭발 효과 만들기 위해 Explosion 클래스 제작. (새 탭)

```
class Explosion {
  float x, y; // 폭발 위치 지정하는 용도임.
  float size = 0; //폭발 크기 지정해줌. 초기값은 0으로 지정 후 age흐르면 증가하
```

게.

```
  float maxSize = 150; //폭발의 최대 크기 지정해줌.
  float lifespan = 30; //폭발의 지속 시간 지정해줌. 30 값 끝나면 폭발 사라짐.
  float age = 0; //폭발의 경과 시간 지정해줌.


  Explosion(float x, float y) {
    this.x = x;
    this.y = y;
  }


//점점 커지는 모션을 주기 위해 챗gpt를 활용함.
  void update() {
    age++; //폭발 경과 시간 흐르게 해줌.
    if (size < maxSize) {
      size += 3; //폭발의 크기가 증가하게 해줌.
    }
  }
```

```
  void display() {
    noFill();
    //stroked마마지막 라인의 투명도(255 - (ge / lifespan) * 255)는
age를 기준으로 점점 사라지도록 설정함. 챗gpt 활용.
    stroke(255, 100, 0, 255 - (age / lifespan) * 255);
    strokeWeight(3);
    ellipse(x, y, size, size);
  }

//폭발하면 어레이리스트에서 제거하기 위해 챗gpt를 활용함.
  boolean isFinished() {
    return age > lifespan; //age가 lifespan보다 커지면 폭발 지속 시간 끝나서 반
환되도록 함.
  }
}
```

Explosion 클래스 메인 코드에 추가.

```
ArrayList<Explosion> explosions; //폭발 효과 관리하는 어레이리스트.



void setup() {

explosions = new ArrayList<Explosion>();

}



void draw() {

if (aEnemy.isHit(m)) {

//하단에 추가하니 실행이 안되어서 if (aEnemy.isHit(m)) {} 구간에 충돌 발생 시 충돌된 적 우주선 위치에 폭발 효과 발생하는 코드 문제 찾아 넣어줌. 챗gpt활용.

            explosions.add(new Explosion(aEnemy.x, aEnemy.y));

}



//폭발효과 만든거 불러옴.

   for (int i = explosions.size() - 1; i >= 0; i--) {

      Explosion aExplosion = explosions.get(i);

      aExplosion.update();
```

```
        aExplosion.display();



        //폭발 효과가 끝나면 제거되도록 하는 코드.
        if (aExplosion.isFinished()) {
            explosions.remove(i);
        }
    }
```

---

7. 화면 좌측 상단에 점수와 ArrayList 크기 기재

- 화면 좌측 상단에 각 ArrayList 크기를 기재함.

메인 코드. ArrayList 값 나타내기 위해 챗gpt활용.

```
// 텍스트 사용+ Arraylist 크기 화면 좌측 상단에 출력하기 위해 챗 gpt 활용함.
void draw() {
  fill(255, 0, 0);
  textSize(22);
  text("Missile Count: " + missiles.size(), 20, 30);

  fill(255, 0, 0);
  textSize(22);
  text("Enemy Count: " + enemies.size(), 20, 60);

  fill(255, 0, 0);
  textSize(22);
  text("Explosion Count: " + explosions.size(), 20, 90);

  // 스코어는 미사일과 적 우주선이 충돌할 때 마다 100 씩 증가함.
  fill(255, 0, 0);
  textSize(22);
  text("Score: " + score, 20, 120);
```

```
}
```