# P3: Threads (Asteroids)*

Abgabetermin: 12. Mai 2024

Punktzahl: 40 + 5* + 5 (Einhaltung der Coding-Richtlinien)

In diesem Projekt wird primär das Thema Threads behandelt. Hierzu sollen Sie einen Clone zum Arcade-Spiel Asteroids[1] entwickeln.

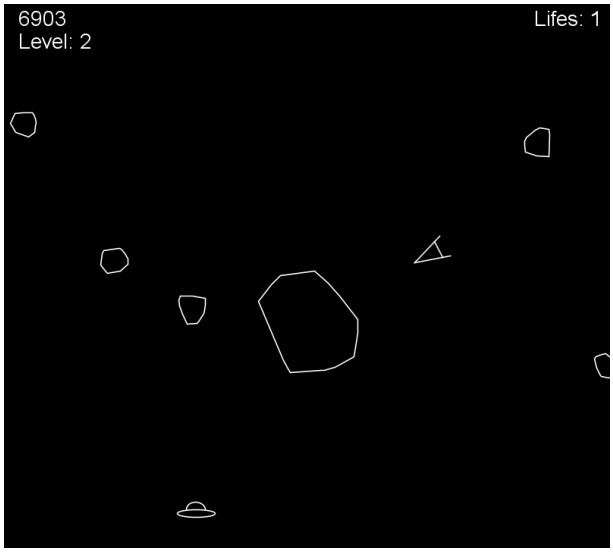
Sie finden in ILIAS die Archivdatei `p3-vorgaben.zip` vor, die Sie in *eclipse* als neues Projekt importieren können. Ändern Sie dann den Namen des Projekts in <IhrNachname.IhrVorname.P3>

## Spielübersicht

Im Spiel Asteroids steuert man ein Raumschiff durch einen Asteroidenschwarm. Dabei hat der Spieler die Möglichkeit das Raumschiff nach links oder rechts zu drehen, das Raumschiff zu beschleunigen oder ein Projektil abzufeuern. Die Geschwindigkeit des Raumschiffs verringert sich mit der Zeit von selbst. Asteroiden, die von Projektilen getroffen wurden, werden zerstört und der Spieler erhält dafür Punkte. Große Asteroiden werden beim Zerstören in mehrere kleine Asteroiden aufgeteilt, die wiederum abgeschossen werden können. Gelegentlich trifft man auf ein feindliches Ufo, welches ebenfalls abgeschossen werden kann.

*Dieses Projekt wurde in einer Seminararbeit entwickelt

[1] https://de.wikipedia.org/wiki/Asteroids




# Vorgaben

Die Vorgaben definieren bereits eine Grundstruktur, die Sie für Ihre Implementierung verwenden sollen. Darin sind mehrere Packages vorgegeben, die nachfolgend kurz erläutert werden. Einige Funktionen, wie z.B. das Erstellen der Asteroiden und die Kollisionserkennung sind bereits implementiert.

### main

Dieses Package beinhaltet die Klassen `Main` und `Settings`. Die Klasse `Main` stellt die Hauptklasse des Projekts da und sollte zum Testen von Ihnen gestartet werden. Die Klasse `Settings` beinhaltet einige konstante Werte für verschiedene Einstellungen wie z.B. die Größe des Fensters und die maximale Geschwindigkeit des Raumschiffs.

### spaceObjects

Dieses Package beinhaltet zunächst die Klasse `SpaceObject`. Diese stellt eine abstrakte Oberklasse für alle Objekte im Weltraum dar. Sie enthält die aktuelle Position des jeweiligen Objekts in Form von 2D-Koordinaten $(x, y)$, dazu den Winkel des Objekts als Radiant sowie die Geschwindigkeit des Objekts, die durch einen Vektor $(vx, vy)$ repräsentiert wird.

Die Klassen `Spaceship`, `Ufo`, `Missile` und `Asteroid` repräsentieren die Objekte im Spiel und sind von der Klasse `SpaceObject` abgeleitet. Die Methode `generateRandomShape` in der Klasse `Asteroid` generiert eine zufälliges konvexes Polygon, welches für die Form des Asteroiden genutzt wird.

**game**

In diesem Package befinden sich die Klassen, die für die Steuerung des Spielgeschehens zuständig sind:

- Die Klasse `GameFrame` ist für die GUI und das Zeichnen aller im Spiel vorkommenden Komponenten zuständig. Sie beinhaltet eine private (innere) Klasse `GamePanel`, in die die Objekte des Spiels gezeichnet werden. Durch die `paint` Methode der Klasse `GamePanel` werden bereits das Raumschiff, das Ufo, die Asteroiden und die Projektile gezeichnet.

- Die Klasse `GameController` steuert das Spiel und soll alle wichtigen Threads beinhalten. Hier sollen im Laufe des Projekts private Klassen erstellt werden, die alle `Thread`s darstellen.

- In der Klasse `GameState` werden alle wichtigen Objekte und Informationen über den aktuellen Spielzustand definiert. Die Klasse besitzt alle Methoden, die zum Ändern des Spielzustandes gebraucht werden. Darüberhinaus besitzt die Klasse `GameState` die Methode `checkCollisions`, die beim Aufruf überprüft, ob Kollisionen zwischen Asteroiden, Ufos, Projektilen und dem Raumschiff des Spielers vorliegen. Hierzu wird überprüft, ob für die Polygone, die diese Objekte begrenzen, Kollisionen (Überschneidungen) vorliegen, Die Methode `checkCollisions` nutzt dabei die Methode `polygonCollision(SpaceObject o1, SpaceObject o2)`, die zwei `SpaceObject` Instanzen mit Hilfe des Separating Axis Theorem[2] auf Kollisionen prüft.

In den Klassen `GameController` und `GameState` sind eine ganze Reihe von Methoden definiert, die Sie in anderen Klassen benötigen. Um auf diese zuzugreifen, sollten Sie sich jeweils mit Hilfe der Klassenmethode `GameController.getInstance` die `GameController` Instanz besorgen und über diese auf die Methoden dieser Klasse zugreifen. Das `GameState` Objekt erhalten Sie dann über die Methode `getGameState` der Klasse. `GameController`.

# Aufgabenstellung

## Aufgabe 1: Graphische Oberfläche und Eingabe (6P)

## (a) GUI Thread (3P)

Erstellen Sie im GameController eine private innere Klasse mit dem Namen `GuiThread`, welche von der Klasse `Thread` abgeleitet ist. Überschreiben Sie die von der Klasse `Thread` geerbte `run`-Methode so, dass die GUI in regelmäßigen Abständen aktualisiert wird, solange der Thread nicht unterbrochen wurde. Gehen Sie wie folgt vor:

[2] siehe z.B. `http://programmerart.weebly.com/separating-axis-theorem.html`

- Ändern Sie in der Methode zunächst den Namen des Threads in "GUI Thread" um.
- Solange der der Thread nicht unterbrochen ist, wiederholen Sie die folgenden Schritte:
  - Rufen Sie die Methode `updateGui` der aktuellen `GameController` Instanz auf.
  - Anschließend soll der Thread für einen Zeitraum "schlafen", welcher durch die in der Klasse `Settings` vorgegebene Variable `GUI_DELAY` spezifiziert ist. Achten Sie darauf, dass die `InterruptedException` abgefangen wird und dort die Methode `interrupt` nochmals aufgerufen wird (um das Flag wieder zu setzen).
- Fügen Sie der Klasse `GameController` eine private Variable der Klasse `GuiThread` hinzu. Ergänzen Sie die (noch leere) Methode `startThreads` so, dass dieses Objekt initialisiert und der Thread dann gestartet wird.

  Fügen Sie den Aufruf der Methode `startThreads` in den Methoden `launch` und `startNewGame` jeweils am Ende der Methoden hinzu.
- Ergänzen Sie die (noch leere) Methode `interruptThreads` in der Klasse `GameController` so, dass der Thread unterbrochen wird.

  Rufen Sie die Methode am Anfang der Methoden `startNewGame` und `gameOver` auf.

Beim Starten sollte nun das Label "Press space" blinken.

## (b) Tastatureingaben (3P)

Die Steuerung der GUI und die Steuerung im Spiel soll über die Tastatur realisiert werden. Hierzu sind in der Klasse `GameState` bereits einige boole'sche Variable deklariert, die das Auftreten ausgewählter Tatstaturereignisse speichern und die bei einem entsprechenden Tastendruck gesetzt werden müssen.

Erstellen sie in der Klasse `GameFrame` eine private innere Klasse `CustomKeyListener`, die das Interface `KeyListener` implementiert und definieren Sie die Methoden des Interface (`keyTyped`, `keyPressed`, `keyReleased`) in der Klasse. Fügen Sie dem `GameFrame` eine Instanz der neuen Klasse über die Methode `addKeyListener` im Konstruktor der Klasse `GameFrame` hinzu. Implementieren Sie nun noch die Methoden `keyPressed` und `keyReleased`. Für die Steuerung des Spiels ist `keyTyped` nicht relevant.

Die Methode `keyPressed` soll Folgendes leisten:

- Überlegen Sie sich eine (noch nicht belegte) Taste, die beim Drücken das Spiel schließt. Prüfen Sie, ob diese verwendet wurde und beenden Sie in diesem Fall das Programm.

- Wenn das Spiel nicht aktiv ist, soll beim Drücken der Taste "space" die Methode `startNewGame` im GameController aufgerufen werden.
- Wenn das Spiel aktiv ist
  - soll die Taste "escape" die Methode `togglePause` aufrufen, sofern die Anzahl der Leben vom Spieler über 0 sind.
  - Die Taste "r" soll `startNewGame` der `GameController` Instanz aufrufen.
- Wenn das Spiel aktiv und nicht pausiert ist, soll die Lenkung des Raumschiffs möglich sein: Wenn die linke, rechte oder obere Pfeiltaste bzw. die Leertaste gedrückt wird, sollen über die Methoden `leftPressed`, `rightPressed` oder `upPressed` bzw. `spacePressed` die zugehörigen boole'schen Variablen in der Klasse `GameState` auf `true` gesetzt werden.

In der Methode `keyReleased` sollen die boole'schen Werte für die Lenkung des Raumschiffs analog zum letzten Punkt für die Methode `keyPressed` wieder auf `false` gesetzt werden.

Hinweise:

- Verwenden Sie die Methode `getKeyCode` auf dem KeyEvent Objekt, welches den Methoden des Listeners übergeben wird, um die jeweils gedrückten Tasten zu ermitteln.
- Verwenden Sie das (aktuelle) `GameState`-Objekt, um über die Instanzmethoden `pressXXX` bzw. `releaseXXX` die boole'schen Variablen aus der Klasse `GameState` zu setzen bzw. deren Wert zu ermitteln. Das `GameState`-Objekt erhalten Sie dabei mit Hilfe der `GameController` Instanz, welche durch das Attribut `gc` referenziert wird.
- Um gerauszufinden, ob das Spiel aktiv oder pausiert ist, verwenden Sie die Instanzmethoden `isActive` und `isPaused` aus der Klasse GameController.

# Aufgabe 2: Spiellogik (12P)

In dieser Aufgabe sollen Sie zunächst alle notwendigen Methoden implementieren, um die- Objekte im Spiel zu bewegen. Danach sollen sie einen Thread schreiben, der diese Methoden aufruft.

## (a) Raumschiff bewegen (7P)

Als Erstes müssen Methoden in der Klasse `Spaceship` implementiert werden, mit denen sich das Raumschiff steuern lässt. Schauen Sie sich hierzu vorher nochmal die Oberklasse `SpaceObject` an, deren Attribute und Methoden Sie hierzu benötigen.

- Implementieren Sie die Methode `rotateLeft`, die den Wert des Radianten des Raumschiffs um `SHIP_ROTATIONSPEED` aus der `Settings-Klasse` verringert. Implementieren Sie analog die Methode `rotateRight`.

- Implementieren Sie die Methode `accelerate`, die die Geschwindigkeit des Raumschiffs erhöhen soll. Jedes `SpaceObject` verfügt über die Attribute `vx` und `vy`, die einen Vektor(vx, vy) darstellen. In der Methode soll der Wert `vx` sich um das Produkt aus der `SHIP_ACCELERATION` aus den Settings und `cos(radiant)` verändern. Der Wert `vy` soll sich analog mit `sin(radiant)` verändern.

- Ergänzen Sie die Methode `move`, die die Position des Raumschiffs um den Vektor(vx,vy) verschieben soll:

  - Rechnen sie zunächst die Länge des Vektors (mit der Formel $\sqrt{v_x^2 + v_y^2}$) aus. Ist die Länge größer als der Wert der Variablen `SHIP_DECELERATION`, soll die Geschwindigkeit des Raumschiffs verringert werden. Verringern sie dazu `vx` und `vy` jeweils um ihren prozentualen Anteil an der Länge des Gesamtvektors, multipliziert mit dem Wert von `SHIP_DECELERATION`.
  - Ist die Länge des Vektors kleiner als `SHIP_DECELERATION` sollen `vx` und `vy` auf 0 gesetzt werden.
  - Prüfen Sie nun, ob die Länge des Vektors größer ist als die maximal erlaubte Länge, die in den Settings als `MAX_SPEED` vorgegeben ist. Wenn die Länge die maximale Länge überschreitet, sollen `vx` und `vy` auf die maximale Länge gesetzt werden.
  - Nun sollen die x und y Werte verschoben werden, in dem Sie den Vektor zu den Koordinaten addieren.

- Implementieren sie nun die Methode `checkBoundaries` in der Klasse `SpaceObject`, die das Raumschiff beim Rausfliegen auf einer Seite auf die andere Seite setzen soll. Überprüfen Sie hierzu die Positionsvariablen $x$ und $y$ des Objekts. Wenn diese nicht mehr in dem Koordinatenbereich sind, der durch das GUI-Fenster definiert wird (also in dem Rechteck `(0,0,Settings.WIDTH, Settings.HEIGHT)`, ändern Sie die Position so, dass das betreffende Objekt an den anderen Rand versetzt wird (rechts <-> links, bzw. oben <-> unten).

  Rufen Sie die Methode `checkBoundaries` am Ende von `move` auf. Zum Schluss soll in `move` die Methode `setShape` aufgerufen werden, um die Form des Raumschiffs an die richtige Stelle zu verschieben.

Alternativ zu den ersten drei Punkten dürfen Sie die Methode auch so implementieren, dass immer ein leichtes Abbremsen des Raumschiffs (durch ein Verringern der Geschwindigkeit) bewirkt wird. Die Geschwindigkeit sollte dabei nicht den Wert 0 unterschreiten. Ergönzen Sie die Methode accelerate in diesem Fall so, das die maximale Geschwindigkeit (durch `Settings.SHIP_MAXSPEED` definiert) nicht überschritten wird.

## (b) Game Thread (5P)

- Erstellen Sie im `GameController` eine private Unterklasse der Klasse `Thread` mit dem Namen `GameThread`. Dieser Thread soll die `SpaceObjects` bewegen und sie auf Kollisionen überprüfen. Implementieren Sie die `run`-Methode wie folgt: Setzen Sie zunächst einen passenden Namen für den Thread und legen Sie eine lokale Variable an,die Sie mit der (aktuellen) `GameController` Instanz initialisieren. Solange dieser Thread nicht unterbrochen wurde, wiederholen Sie folgende Schritte in einer Schleife:
  - Rufen Sie die Methode `checkCollisions` auf dem im `GameController` gespeicherten `GameState` auf.
  - Danach soll die Methode `moveObjects` aufgerufen werden.
  - Sofern das Spiel aktiv ist, soll der Score des Spielers um 1 erhöht werden.
  - Anschließend soll der Thread für den Zeitraum schlafen, welcher durch die Variable `currentDelay` definiert wird.
- Implementieren Sie die in der Klasse `GameState` leer vorgegebenen Methoden `moveShip`, `moveMissiles` und `moveAsteroids`. Hnweis: Um nicht mehr auf dem Spielfeld sichtbare Asteroiden und Projektile aus den zugehörigen Listen zu löschen, sollten Sie Sie einen Iterator verwenden, mit dem Sie diese Listen durchlaufen und, in Abhängigkeit einer passenden Bedingung, einzelne Elemente löschen können.
  - In `moveShip` sollen die Methoden aufgerufen werden, die sie in Aufgabe (2a) für die Bewegung des Schiffs implementiert haben (`rotateLeft`, `rotateRight`, `accelerate`), sofern die zugehörigen boole'schen Variablen den Wert `true` haben.
  - Für die Methode `moveMissiles` müssen Sie zunächst die Methode `move` in der Klasse `Missile` vervollständigen. Aktualisieren sie dazu die Punkte x und y (indem Sie die Werte aus dem Bewegungsvektor darauf addieren) und rufen sie am Ende der Methode `setBoundingBox` auf.
  - Ergänzen sie dann die Methode `moveMissiles` so, dass für jedes Objekt in der Liste `missiles` geprüft wird, ob sich die Missile noch im Spielfeld befindet. Falls Sie außerhalb ist, soll sie aus der Liste entfernt werden. Rufen Sie dann für jede Missile aus der Liste die Methode `move` auf. Wiederholen sie die beiden Schritte für die Ufos in der Liste `missilesUfo`. Hinweis: Um zu überprüfen, ob ein Objekt nicht mehr im Bereich des Spielfelds ist, verwenden Sie die Position des Objekts sowie dessen Breite und Höhe und die Höhe und Breite des Spielfelds.
  - Ergänzen Sie `moveAsteroids` analog zu `moveMissiles`. In der Methode `move` in der Klasse Asteroid muss am Ende `setShape` aufgerufen werden, um die Form an die richtige Stelle zu verschieben. Entfernen Sie Asteroiden nur,

wenn die Position des Asteroiden bereits mindestens 100 Pixel oberhalb oder unterhalb der Grenzen des Spielfelds liegt.

- Rufen Sie in der Methode `moveObjects` nacheinander die Methoden `moveShip`, `moveMissiles` und `moveAsteroids` auf. Die Methode `shootMissile` soll darüberhinaus aufgerufen werden, wenn das Attribut `spacePressed` den Wert `true` hat. Falls ein Ufo im Spiel existiert (mit der Methode `ufoActive` prüfen), soll zusätzlich die Methode `moveUfo` aufgerufen werden.

- Definieren Sie in der Klasse `GameController` ein `GameThread` Attribut. Ergänzen Sie die Methoden `startThreads` und `interruptThreads` so, dass dieses Attribut dort mit einer Instanz der Klasse `GameThread` initialisiert wird bzw. der Thread unterbrochen wird.

Beim Starten der Applikation sollte man nun das Raumschiff steuern können.

# Aufgabe 3: Asteroiden (12P)

Die Asteroiden sollen immer außerhalb des Spielfelds spawnen und dann durch das Spielfeld durchfliegen. Sobald ein Asteroid wieder aus dem Spielfeld geflogen ist, soll er wieder aus dem Spiel entfernt werden.

Die Methode `generateRandomConvexPolygon(int n)` in der Klasse `Asteroid` erstellt hierzu bereits ein konvexes Polygon mit `n` Kanten.

## (a) Konstruktor Asteroiden (5P)

Zunächst sollen Sie den Default-Konstruktor der Klasse Asteroid vervollständigen, welcher zufallsbasiert die Größe, die initiale Position sowie die Geschwindigkeit des neuen Asteroiden initialisiert:

- Die Klasse Asteroid besitzt ein Enum mit verschiedenen Größen für die Asteroiden. Setzen Sie die `size` des Asteroiden im Konstruktor auf eine zufällige Größe aus dem Enum.

- Setzen Sie dann die `x` und `y` Werte des Asteroiden zufällig auf einen Wert außerhalb des Spielfelds, also innerhalb der Quadranten, die entweder links, rechts oberhalb oder unterhalb des Spielfelds liegen. Hinweis: Sie können dazu die Punkte der vier Linien nutzen, die einen Abstand von 50 zu den entsprechenden Rändern des Spielfelds haben ($x = 50$, $x = -50$, $y = 50$, $y = -50$) Für den Quadranten oberhalb des Spielfeldes sollten die x Werte zwischen 0 und `Settings.WIDTH` gewählt und y auf $-50$ gesetzt werden.

- Der Radiant der Asteroiden muss so gesetzt werden, dass die Asteroiden immer in Richtung Spielfeld fliegen.
  Hinweis: Ein Asteroid mit dem Radianten 0 (bzw. $0.5*$ `Math.Pi`, `Math.Pi`, $1.5*$

`Math.Pi`, 2∗ `Math.Pi`) fliegt nach rechts (bzw. oben, links, unten, rechts). Der Radiant soll immer einen zufälligen Wert im Bereich eines halben Kreises annehmen. Für Asteroiden, die oberhalb des Spielfeldes spawnen, wären das Werte zwischen 0 und `Math.Pi` (da die y-Achse nach unten zeigt).

- Initialisieren Sie nun den Geschwindigkeitsvektor: Generieren sie hierzu einen zufälligen `double` Wert zwischen 2 und 4. Setzen Sie `vx` auf das Produkt zwischen der Variable und `cos(radiant)`. Setzen Sie `vy` auf das Produkt zwischen der Variable und `sin(radiant`).

- Rufen Sie zum Schluss die Methode `generateRandomShape` auf. Sie setzt die Form des Asteroiden auf ein konvexes Polygon mit `n` Kanten, `n` hängt dabei von der Größe des Asteroiden ab.

## (b) Asteroids Thread (2P)

Erstellen Sie in der Klasse `GameController` eine private Unterklasse `SpawnThread` der Klasse `Thread`. Gehen Sie dazu analog zu Aufgabe (1a) und (1b) vor, indem Sie die `run`-Methode überschreiben, ein Attribut der neuen Klasse in der Klasse `GameController` anlegen, dieses in der Methode `startThreads` initialisieren und den Thread starten sowie den Thread in der Methode `interruptThreads` unterbrechen.

Der Thread soll Folgendes leisten: Solange der Thread nicht unterbrochen wurde, soll geprüft werden, ob die Anzahl der Asteroiden in der Liste `asteroids` kleiner ist als der Wert des Attributs `currentMaxAsteroids`. Die Liste `asteroids` ist dabei in der Klasse `GameState` definiert. Ist dies der Fall, soll der Liste mit Hilfe der Instanzmethode `spawnAsteroid` (aus der Klasse `GameState`) ein neuer Asteroid hinzugefügt werden. Anschließend soll der Thread für die in der Klasse `Settings` definierte Zeit `ASTEROIDS_SPAWNRATE` schlafen.

## (c) Aufteilen der Asteroiden (5P)

Wenn ein großer Asteroid abgeschossen wird, soll er sich in mehrere kleine Asteroiden aufteilen. Erstellen Sie hierzu in der Klasse `Asteroid` in eine Methode `splitAsteroid`, die eine Liste mit neuen Asteroiden zurückgibt:

- Wenn das Attribut size nur TINY beträgt, soll der Asteroid sich nicht aufteilen. Geben Sie in diesem Fall eine leere Liste zurück. Bei jeder anderen Größe soll sich der Asteroid in eine bestimmte Anzahl an kleinen Asteroiden aufteilen:

  - Beträgt die Größe des aktuell betrachteten Asteroiden `SMALL`, soll der Asteroid sich in zwei Asteroiden der Größe `TINY` aufteilen.
  - Beträgt die Größe `NORMAL`, soll sich der Asteroid in einen Asteroiden der Größe SMALL und einen weiteren Asteroiden der Größe `TINY` aufteilen.

  - Beträgt die Größe `LARGE`, soll der Asteroid sich in einen Asteroiden der Größe NORMAL, einen der Größe `SMALL` und einen Asteroiden der Größe `TINY` aufteilen.

- Die neuen Asteroiden sollen die Geschwindigkeit des alten Asteroiden, also die Länge des Vektors, behalten. Allerdings soll sich die Richtung der Asteroiden um 22,5° nach oben und um 22,5° nach unten ändern (also um $\pm\frac{\Pi}{8}$). Im Fall `Large` soll der `NORMAL` Asteroid die Richtung des alten Asteroiden beibehalten. Berechnen Sie die Komponenten `vx` und `vy` des Geschwindigkeitsvektor auf der Basis des angepassten Winkels neu.

  Mit dem Winkel $r = \frac{\Pi}{8}$ und dem (bisherigen) Geschwindigkeitsvektor $(v_x, v_y)$ kann dieser wie folgt neu berechnet werden:

  nach oben

  $$v_x = v_x * \cos(r) - v_y * \sin(r)$$
  $$v_y = v_x * \sin(r) + v_y * \cos(r)$$

  nach unten

  $$v_x = v_x * \cos(-r) - v_y * \sin(-r)$$
  $$v_y = v_x * \sin(-r) + v_y * \cos(-r)$$

- Setzen Sie die Positionen der Bruchstücke auf Werte, die im Vergleich zu den Werten des ursprünglichen Asteroiden leicht verändert sind.

- Verwenden Sie den vorgegebenen privaten Konstruktor der Klasse `Asteroid` um neue Asteroiden zu erstellen.

Entfernen Sie nun in der Klasse `GameState` in der Methode `checkCollisions` jeweils die Kommentare in den Zeilen, in denen die Methode `splitAsteroid` aufgerufen wird.

# Aufgabe 4: Zusätzliche Threads (10P+5*P)

In dieser Aufgabe sollen Sie nun noch die Klasse `GameController` um einige weitere Threads erweitern.

## (a) Ufo Thread (3P)

Hier sollen Sie einen Thread implementieren, der ein Ufo erscheinen lässt und bewegt. Das Ufo soll sich zufällig nach oben und unten bewegen. Dabei bleibt der `vx` Wert immer gleich und der `vy` wird zufällig geändert.

- Ergänzen Sie zunächst die Klasse `Ufo` durch eine parameterlose Methode mit dem Namen `changeVelocity`, welche keinen Wert zurückgibt. In der Methode soll mit einer Wahrscheinlichkeit von je 50% der Wert des Attributs `vy` auf 2 bzw. -2 gesetzt werden.
- Erstellen Sie nun in der Klasse `GameController` eine private innere Klasse `UfoThread`. Der Thread soll solange laufen bis er abgebrochen wird.
- Prüfen Sie über die Methode `isActive` im `GameController`, ob das Spiel aktiv ist. Falls es aktiv ist, soll der Thread für `UFO_SPAWNRATE` schlafen.
- Prüfen Sie mit Hilfe der Methode `ufoActive` (aus der Klasse `GameState`), ob bereits ein Ufo existiert. Wenn (noch) kein Ufo existiert, soll die Methode `spawnUfo` aufgerufen werden und der Thread für 1000 ms schlafen.
- Anschließend soll in einer Schleife, die solange laufen soll, bis das Ufo nicht mehr aktiv ist, Folgendes gemacht werden:
  - Zunächst soll der Thread für 400 ms schlafen.
  - Dann soll die Methode `changeVelocity` auf dem Ufo aus dem GameState aufgerufen werden.
  - Lassen Sie den Thread dann erneut für 200 ms schlafen.
  - Dann soll zufällig zu 20% ein Projektil abgeschossen werden. Rufen Sie dazu die Methode `shootMissileUfo` im `GameState` auf.

Starten Sie den Thread in `startThreads` und brechen Sie ihn beim Aufruf von `interruptThreads` wieder ab.

## (b) Invincibility Thread (4P)

Der Spieler startet das Spiel mit drei Leben. Wird er getroffen, sollen seine Leben um 1 verringert werden. Sobald das Raumschiff mit etwas kollidiert, soll es für einen festen Zeitraum blinken und von den Kollisionen ausgeschlossen sein. Das Spiel endet, sobald die Anzahl der Leben des Spielers auf 0 gesunken ist.

- Erstellen Sie eine private Klasse `InvincibilityThread` in der Klasse `GameController`. Der Thread soll Folgendes leisten:
  - Der Thread soll für die in Settings angegebene Zeit `SHIP_INVINCIBLE_TIME` laufen oder bis er von außen abgebrochen wird. Hinweis: Definieren Sie sich ein Attribut in dieser Klasse, welches die Restzeit definiert (und mit dem Wert von `SHIP_INVINCIBLE_TIME` initialisiert wird) und reduzieren Sie den Wert des Attributs mit jeder "Schlafphase" des Threads um den entsprechenden Wert. Die Laufzeit für den restlichen Code können Sie dabei ignorieren.

  - Zu Beginn (jedes Schleifendurchlaufs in der `run`-Methode) soll die Methode `setInvincible` aufgerufen werden, um das Schiff unverwundbar zu machen. Am Ende soll die Methode `setVulnarable` aufgerufen werden, um es wieder verwundbar zu machen. Dazwischen soll über die Methode `setShipAlphaValue` die Transparenz des Raumschiffs alle 10 ms in einem Intervall von 0 bis 255 nach oben und wieder nach unten geändert werden.

- Ergänzen Sie die Methode `shipGotHit` in der Klasse `GameState`: Zunächst soll die Anzahl der Leben über den Aufruf der Methode `loseLife` verringert werden. Wenn danach die Anzahl an Leben 0 beträgt, soll die Methode `gameOver` vom GameController aufgerufen werden. Anderenfalls soll die Position des Schiffs (Attribut `spaceShip`) mit Hilfe der Methode `resetPosition` wieder auf die Mitte gesetzt werden. Danach soll der `InvincibilityThread` mit Hilfe der Methode `startInvincibleThread` (aus der Klasse `GameController`) gestartet werden.

Der Thread soll in der Methode `interruptThreads` unterbrochen werden, allerdings nicht beim Start des Spiels gestartet werden.

## (c) Difficulty Thread (3P)

Im Original besitzt das Spiel mehrere Level. Diese Version soll in einem Endlosmodus laufen, allerdings soll mit der Zeit der Schwierigskeitsgrad erhöht werden. Dabei soll das Spiel schneller werden und die Anzahl an Asteroiden soll sich erhöhen.

- Erstellen Sie in der Klasse `GameController` eine neue innere Klasse mit dem Namen `IncreaseDifficultyThread`, die von der Klasse Thread abgeleitet ist. Der Thread soll solange laufen, bis er von außen abgebrochen wird.

- Wenn der Thread gestartet wurde, soll dieser für die in der Klasse `Settings` durch den Wert von `INCREASE_DIFFICULTY_DELAY` vorgegebene Zeit schlafen. Rufen Sie dann die Methode `increaseLevel` im `GameState` auf, um das Label in der GUI auf das neue Level zu aktualisieren. Reduzieren Sie den `currentDelay` um 1, solange er noch über 10 ist. Erhöhen Sie `currentMaxAsteroids` um 1, solange die Anzahl noch unter 40 ist. Die Variablen `currentDelay` und `currentMaxAsteroids` sind als Attribute in der Klasse `GameController` .

- Optional können Sie die Leben des Spielers z.B. alle zwei Level um 1 erhöhen.

Starten Sie den Thread in der Methode `startThreads` und brechen Sie ihn über die Methode `interruptThreads` wieder ab.

## (d*) Threads pausieren (5*P)

Das Spiel soll über die Taste *Escape* pausiert und wieder fortgesetzt werden können. Dazu müssen alle Threads (`GuiThread`, `GameThread`, `SpawnThread`, `UfoThread`, `InvincibleThread`

und `IncreaseDifficultyThread`) pausiert werden können. Hierzu sollen Sie einen Mechanismus implementieren, bei dem die Threads über die `wait`-Methode (aus der Klasse `Object`) in den Zustand "nicht ausführbar" versetzt werden und über die Methode `notifyAll` (aus der Klasse `Object`) wieder in den Zustand "ausführbar".

Da dieser Mechanismus für alle Klassen der gleiche ist, sollen Sie zunächst eine neue, von `Thread` abgeleitete Klasse `PausableThread` schreiben und dann alle o.g. Thread-Klassen von `PausableThread` ableiten (anstelle von `Thread`). Anschließend müssen Sie noch einige Methodeaufrufe ergänzen.

Gehen Sie daher wie folgt vor:

- Definieren Sie in der Klasse `GameController` eine neue innere Klasse mit dem Namen `PausableThread`, die Sie wie folgt ausstatten:
  - Definieren Sie ein (privates) boole'sches Attribut mit dem Namen `paused`.
  - Schreiben Sie eine öffentliche, parameterlose Methode `pause`, die dieses Attribut auf den Wert `true` setzt.
  - Erstellen sie ein (privates) Attribut vom Typ `Object`) mit dem Namen `lock`.
  - Fügen Sie der Klasse eine protected Methode `lock` hinzu. In dieser Methode soll auf dem Objekt die Methode `wait` aufgerufen werden, falls das Attribut `paused` den Wert `true` hat.
  - Erstellen sie eine öffentliche Methode mit dem Namen `continueThread`. Sie soll den Wert von `paused` wieder auf `false` setzen und die Methode `notifyAll` auf dem Objekt `lock` aufrufen.
  - Ergänzen Sie die Methoden noch so, dass Methodenaufrufe auf dem `lock` Objekt auf diesem Objekt synchronisiert werden. und fangen Sie, wo erforderlich, die `InterruptedException` ab. Rufen Sie dann die `interrupt` Methode auf dem aktuellen Thread auf, um so das `InterruptedFlag` neu zu setzen.
- Ändern Sie die Oberklasse der o.g. Thread-Klassen auf die Klasse `PausableThread` und ergänzen Sie die `run`-Methoden der Thread-Klassen an passenden Stellen durch den Aufruf der `lock`-Methode.
- Ergänzen Sie die Methode `togglePause` im `GameController` wie folgt:
  - Falls das Spiel pausiert ist (erkennbar am Wert des Attributs `gamePaused` der Klasse `GameController`), setzen Sie mit Hilfe der Methode `continueThread` alle Threads fort.
  - Ist das Spiel aktuell nicht pausiert, pausieren Sie alle o.g. Threads mit Hilfe der Methode `pause`.
  - Ändern Sie den Wert des Attributs `gamePaused` mit jedem Aufruf der Methode `togglePause` von `true` auf `false` bzw. umgekehrt.

Das Spiel sollte sich nun über die *Escape* Taste pausieren und fortsetzen lassen.

# Allgemeine Hinweise

- Greifen Sie auf die `GameController` Instanz von außerhalb (dieser Klasse) immer über die Klassenmethode `GameController.getInstance` zu.
- Alle Threads erben von der Klasse `Thread` und gehören als private innere Klassen in die `GameController` Klasse. Die Aufgaben der Threads soll in der überschriebenen `run` Methode implementiert werden.
- Versehen Sie alle Methoden mit dem Modifier `synchronized`, bei denen es nötig ist. Sie können ggf. auch Blöcke in einer Methode auf einem Objekt synchronisieren.
- Der Radiant gibt die Richtung an, in die die Objekte zeigen. Bei 0 zeigt das Objekt nach rechts, bei $0.5\Pi$ nach unten, bei $\Pi$ nach links und bei $1.5\Pi$ nach oben. Bei $2\Pi$ zeigt es wieder nach rechts.
- Die Einstellungen in der `Settings` Klasse dürfen sinnvoll verändert werden.
- Das das Interrupt Flag durch die InterruptedException wieder zurückgesetzt wird, müssen Sie die Methode `interrupt` im Catch-Block zu der jeweiligen Exception nochmals für den aktuellen Thread aufrufen.
- Mit Hilfe der Klasse `TestKeyCode`, die Sie ebenfalls auf Ilias finden, können Sie sich die Keycodes mit den zugehörigen Beschreibungen in der Konsole ausgeben lassen. Voreingestellt ist der Bereich $1 - 100$.
- Wenn Sie zufällige Werte in Thread-Klassen benötigen (z.B. in Aufgabe (3a)), verwenden Sie die Klasse `java.util.concurrent.ThreadLocalRandom` um diese zu generieren.