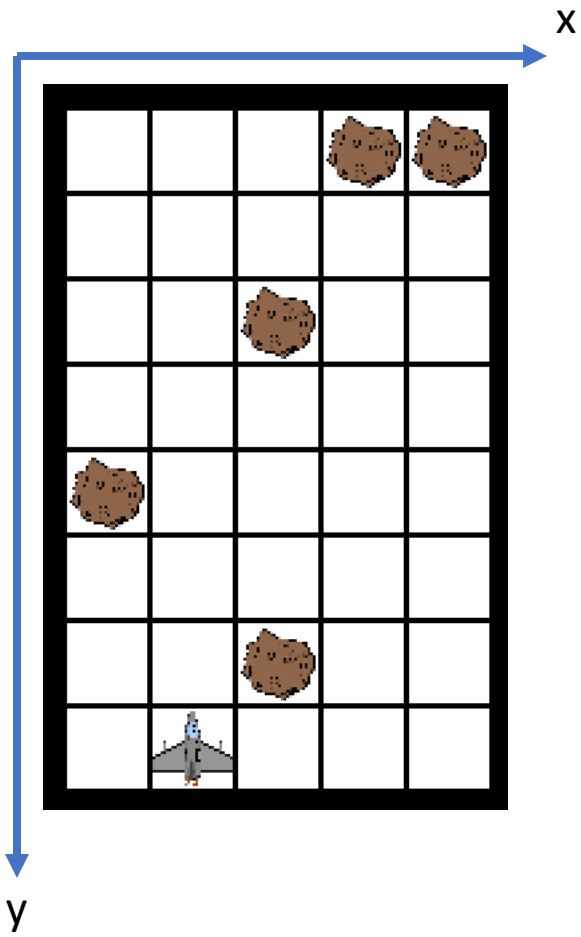
EFIF ConsoleGames: Modellieren

Modell

- «Ein Modell ist ein **abstraktes Abbild der Wirklichkeit**, das *wesentliche Aspekte* enthält, jedoch aufgrund der Reduktion leichter zu untersuchen ist.»
  Johannes Magenheim (2009), Informatik macchiato, S. 58
- Modell für Game:
  - Abbilden des **Spielzustands** (game state) im Code
  - Komplexität der Realität wird auf **wesentliche Faktoren reduziert**
  - Die Realität wird also *nicht* vollständig dargestellt
- **Ziel:** Verstehen, wie man Game optimal modelliert.
- -> anhand von BeiSpiel

Beispiel: Asteroiden Game

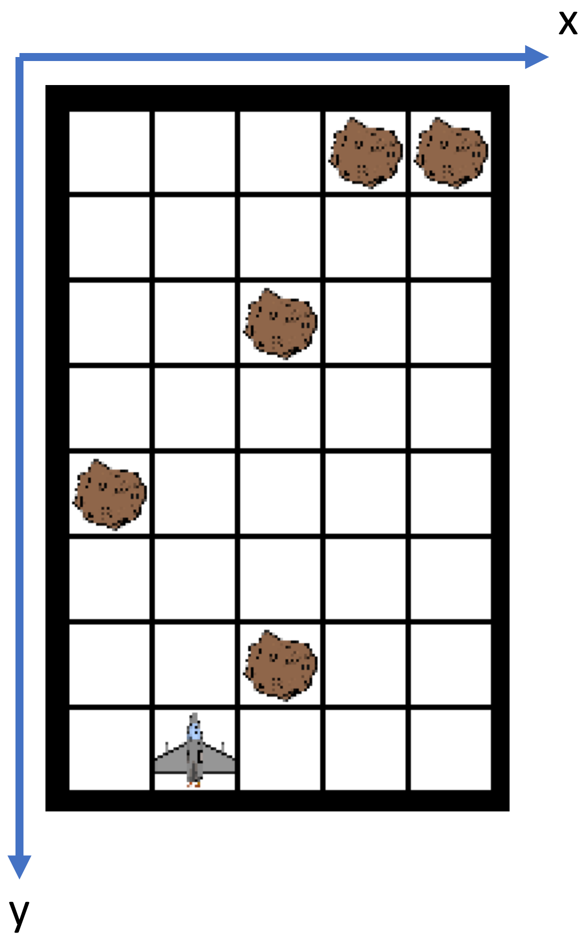
- Regeln:
  - Jeder Asteroid bewegt sich alle 0.5s um 1 Feld nach unten.
  - Asteroiden können alle 0.5s mit gewisser Wahrscheinlichkeit am oberen Rand an zufälliger Position spawnen
  - Player (Jet) muss Asteroiden ausweichen
  - Player Bewegung: links, rechts, (vorne, hinten)
  - Kollision mit Asteroid: Game Over
  - Ziel: Möglichst lange überleben
- Ziel: Modell für Game überlegen



Asteroiden Game (Modell 1)

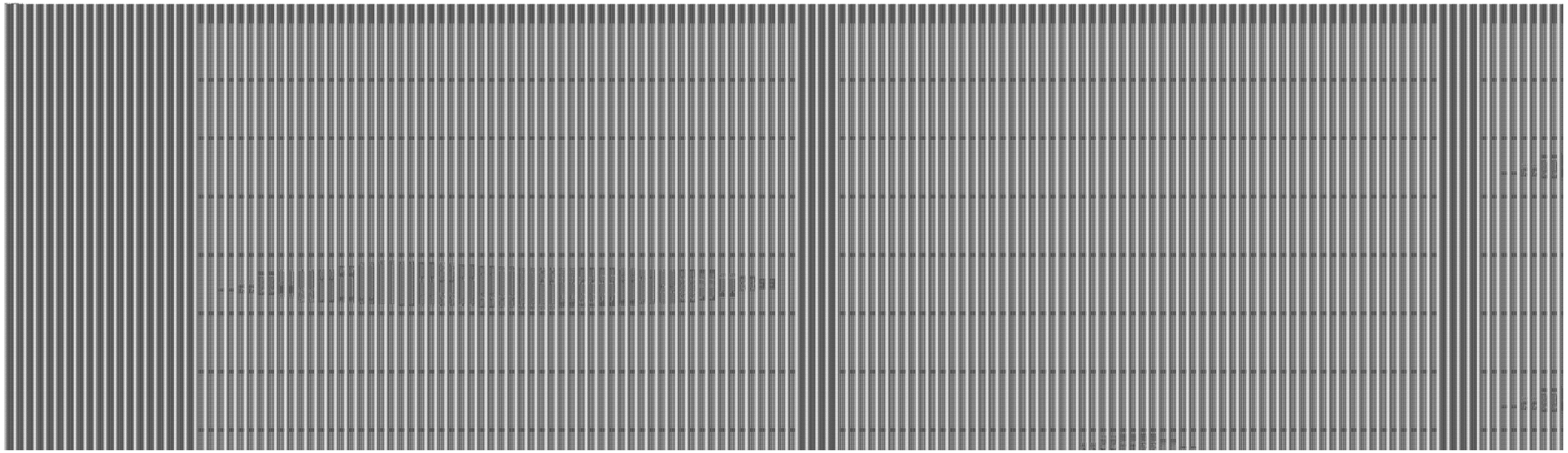
- Wirklichkeit -->
- Wie bilden wir diese im Spiel ab?
- **Modell 1**: Möglichst nahe an Wirklichkeit
- 2D-Array, jeder Eintrag speichert Farbe von Pixel an Koordinate (x,y)
  - Pixel besteht aus RGB-Wert (Red,Green,Blue)
  - Damit kürzer: Hex-Format (Basis 16)
  - 0xFF0000, 0x00FF00, 0x0000FF, 0x000000, 0xFFFFFF

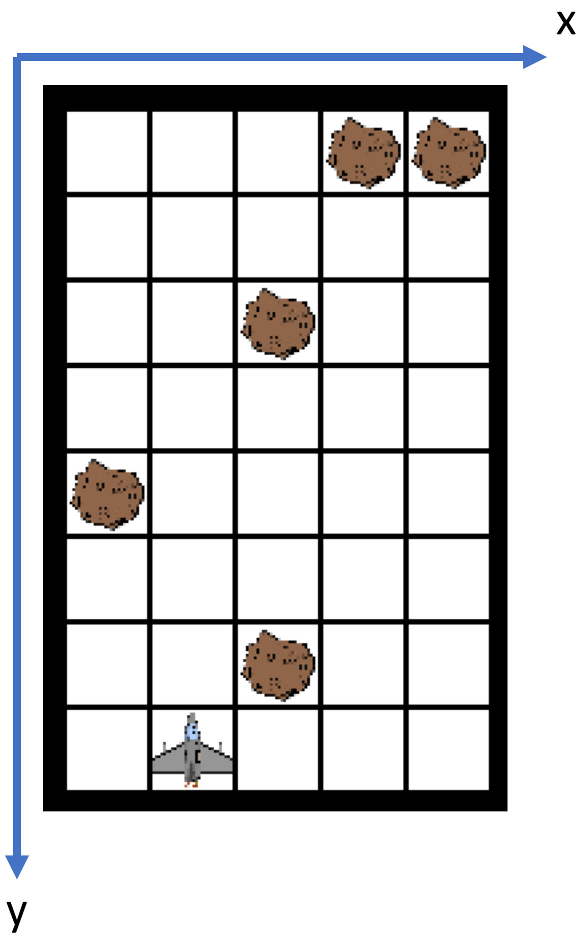
```
int[,] gameState = {
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x0000
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x0000
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x0000
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x0000
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x0000
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x0000
```



Asteroiden Game (Modell 1)

```
int[,] gameState = {
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x00000
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x00000
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x00000
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x00000
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x00000
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x000000","0x00000
```

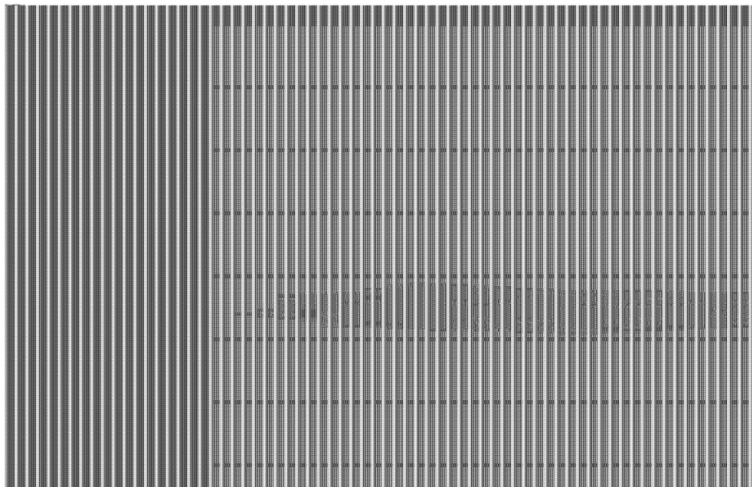




- Probleme mit diesem Modell?

Asteroiden Game (Modell 1)

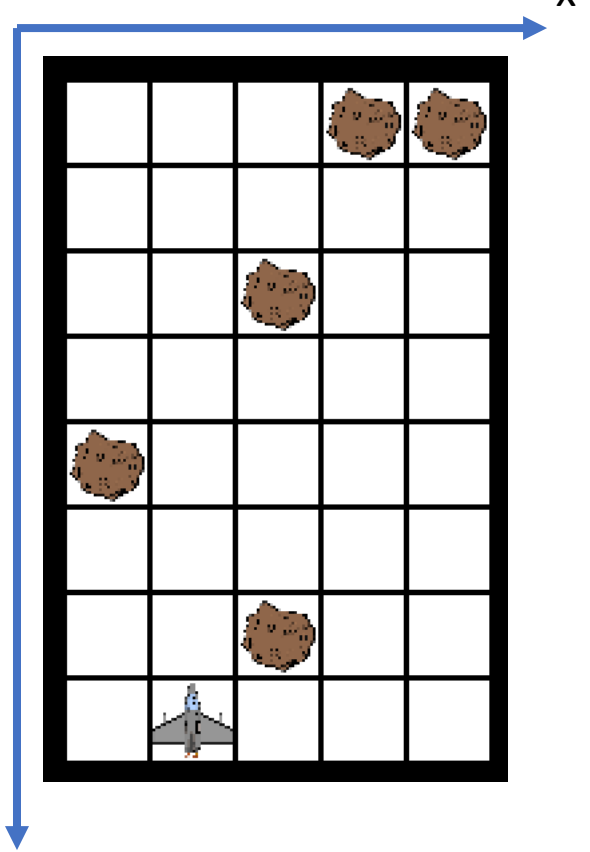
```
int[,] gameState = {
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x0000
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x0000
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x0000
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x0000
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x0000
{"0x000000","0x000000","0x000000","0x000000","0x000000","0x0000
```



- Probleme Speicher:
  - Anzahl Pixel: 546 x 350 = 191’100
  - Pro Pixel 3 Byte (je 1 für R,G und B) -> knapp 600 kByte
- Problem Skalierung
  - Höhere Auflösung -> noch höherer Speicherbedarf
- Problem Spiellogik implementieren:
  - Spielstand updaten sehr umständlich (z.B. Asteroiden um 1 Feld nach unten verschieben, auf Kollisionen überprüfen, …)
- Problem Model vs. View: keine Trennung!
- Kurz: Viel zu kompliziert
- Wollen: Modell mit **viel höherer Abstraktion**
- **Ideen?** Besprecht in Gruppe

Asteroiden Game (Modell 2)

- **Modell 2,** Überlegung: Game hat 5x8 = 40 Felder, reicht, wenn man Information für diese speichert
- -> Abstraktion!
- Wie Felder dann dargestellt werden, ist Problem der View-Funktionen (Stichwort: **Model-View Trennung**)
- Probleme:
  - Speichert Information vieler leerer Felder
  - `gameState` kann ziemlich gross werden für mehr Zellen
  - Spiellogik implementieren immer noch eher mühsam. Bsp. Player ‘p’ nach links verschieben:
    - Player in Array finden (Koordinaten ermitteln)
    - Erst dann kann verschoben werden

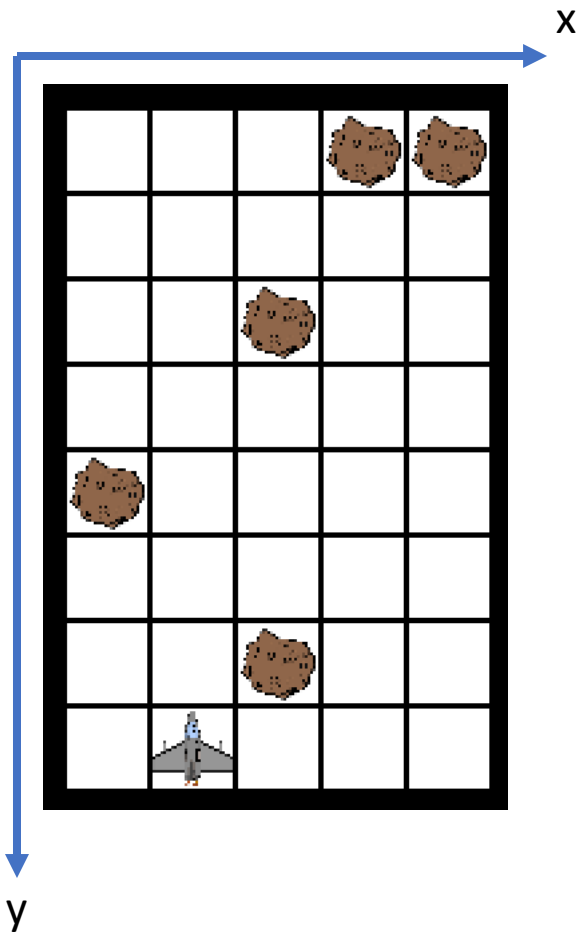


```
char[,] gameState = {
    { ' ', ' ', ' ', 'a', 'a' },
    { ' ', ' ', ' ', ' ', ' ' },
    { ' ', ' ', 'a', ' ', ' ' },
    { ' ', ' ', ' ', ' ', ' ' },
    { 'a', ' ', ' ', ' ', ' ' },
    { ' ', ' ', ' ', ' ', ' ' },
    { ' ', ' ', 'a', ' ', ' ' },
    { ' ', 'p', ' ', ' ', ' ' },
};
```

Asteroiden Game (Modell 3)

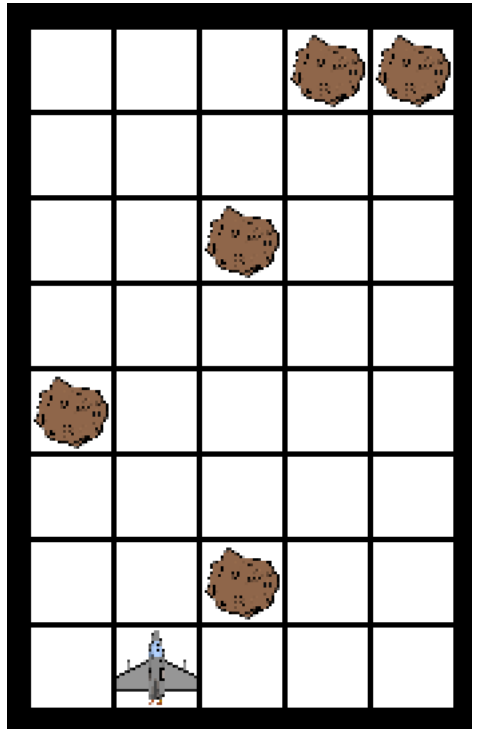
- **Modell 3**, Überlegung: Player und Asteroiden haben Koordinate im Bereich (0,0) [oben links] bis (4,7) [unten rechts]
- Speichere einfach Koordinaten des Players und der existierenden Asteroiden:

```
int[] player = { 7, 1 };
List<int[]> asteroids = new() {
    new[] { 0, 3 },
    new[] { 0, 4 },
    new[] { 2, 2 },
    new[] { 4, 0 },
    new[] { 6, 2 }
};
```



- Maximale Abstraktion!
- Neuer Asteroid: Neues int-Array zu Liste hinzufügen
- Asteroid läuft aus Bildschirm: Aus Liste entfernen

Asteroiden Game



Modell 1

```
int[,] gameState = {
{"0x000000","0x000000","0x000000","0x00000
{"0x000000","0x000000","0x000000","0x00000
{"0x000000","0x000000","0x000000","0x00000
{"0x000000","0x000000","0x000000","0x00000
{"0x000000","0x000000","0x000000","0x00000
{"0x000000","0x000000","0x000000","0x00000
{"0x000000","0x000000","0x000000","0x00000
```

Modell 2

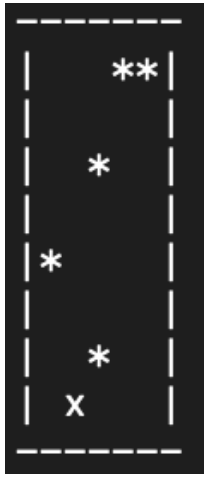
```
char[,] gameState = {
    { ' ', ' ', ' ', 'a', 'a' },
    { ' ', ' ', ' ', ' ', ' ' },
    { ' ', ' ', 'a', ' ', ' ' },
    { ' ', ' ', ' ', ' ', ' ' },
    { ' ', ' ', ' ', ' ', ' ' },
    { 'a', ' ', ' ', ' ', ' ' },
    { ' ', ' ', ' ', ' ', ' ' },
    { ' ', ' ', 'a', ' ', ' ' },
    { ' ', 'p', ' ', ' ', ' ' },
};
```

Modell 3

```
int[] player = { 7, 1 };
List<int[]> asteroids = new() {
    new[] { 0, 3 },
    new[] { 0, 4 },
    new[] { 2, 2 },
    new[] { 4, 0 },
    new[] { 6, 2 }
};
```

Abstraktionslevel

Asteroiden Game



- Ziel: Modell 3 ausarbeiten für Console Game

- **Gruppenarbeit I (**auf *Papier)*:
  Bestimmt & notiert:
  - Benötigte **Variablen:** Deklaration & Erklärung:
    - `int[] player;` // y & x Koordinaten von player
    - `List<int[]> asteroids;` // Liste mit Koords …
    - Weitere Variablen?
  - Benötigte **Funktionen** mit Erklärung und ob Model oder View:
    - `NameOfFunction();` // Erklärung, Model oder View?

Regeln:

- Jeder Asteroid bewegt sich alle 0.5s um 1 Feld nach unten.
- Asteroiden können alle 0.5s mit gewisser Wahrscheinlichkeit am oberen Rand an zufälliger Position spawnen
- Player (Jet) muss Asteroiden ausweichen
- Player Bewegung: links, rechts, vorne, hinten
- Kollision mit Asteroid: Game Over
- Ziel: Möglichst lange überleben

Asteroiden Game

- **Variablen:**
  - `int[] player;` // y & x Koordinaten von player
  - `List<int[]> asteroids;` // Liste mit Koordinaten der Asteroiden
  - `int[2] dim;` // Anzahl Pixel in x- & y-Richtung
    - Alternative: `int nx,ny;` // Anz. Pixel in separaten Variablen
  - `int pSpawn;` //Wahrscheinlichkeit, dass neuer Asteroid spawned (int)
  - (`bool isGameOver;` // ist false, wird true sobald Game Over)
  - `int timeUpdate;` // Zeit in ms, nach der Asteroiden updated werden
- **Funktionen:**
  - `CheckUserInput();` // Überprüft, ob Pfeiltaste gedrückt wurde
  - `MovePlayer();` // Überprüft ob player bewegen soll
  - `MoveAsteroids();` // Bewegt Asteroiden um 1 nach unten
  - `SpawnAsteroid();` // Fügt mit Wahrschk. pSpawn neuen Asteroiden hinzu
  - `CleanUpAsteroids();` // Entfernt Asteroiden, die aus Bild laufen
  - `CollisionCheck();` // Überprüft ob player mit Asteroid kollidiert
  - `Display();` // zeigt Game an

Model-Code **View-Code**

Asteroiden Game

- **Gruppenarbeit II** (auf Papier): Skizziere nun den *Inhalt der play-Methode* im Framework:
  - Deklaration der Variablen
  - Aufruf der Funktionen
  - Game Loop (while (true))
  - Achtung: Funktionen sollen *nicht ausprogrammiert* werden!
  - Ähnlich wie das, was bei HangMan vorgegeben wurde (nur etw. ausführlicher, mit Variablen)

- **Variablen:**
  - `int[] player;` // y & x Koordinaten von player
  - `List<int[]> asteroids;` // Liste mit Koordinaten der Aster
  - `int[2] dim;` // Anzahl Pixel in x- & y-Richtung
    - Alternative: `int nx,ny;` // Anz. Pixel in separaten Variablen
  - `int pSpawn;` //Wahrscheinlichkeit, dass neuer Asteroid spawne
  - (`bool isGameOver;` // ist false, wird true sobald Game Over)
  - `int timeUpdate;` // Zeit in ms, nach der Asteroiden updated w
- **Funktionen:**
  - `CheckUserInput();` // Überprüft, ob Pfeiltaste gedrückt wurd
  - `MovePlayer();` // Überprüft ob player bewegen soll
  - `MoveAsteroids();` // Bewegt Asteroiden um 1 nach unten
  - `SpawnAsteroid();` // Fügt mit Wahrschk. pSpawn neuen Aster
  - `CleanUpAsteroids();` // Entfernt Asteroiden, die aus Bild lau
  - `CollisionCheck();` // Überprüft ob player mit Asteroid kollid
  - `Display();` // zeigt Game an

```
// Variable declarations allowed here
while (true)                        // The game repeats until fin
{
    // Variable declarations allowed here
    ReadSecretWord();               // Player 1: Enter the secret
    HangTheMan();                   // Screen output for a good s
    while (true)                    // Player 2: Make your guesse
    {
        ReadOneChar();              // Handle input of one char.
        EvaluateTheSituation();     // Game Logic goes here
        HangTheMan();               // Screen output goes here
    }
    QuitOrRestart(); // Ask if want to quit or start new game
}
```

Asteroiden Game

```
int[] dim = { 8, 5 };
int[] player = { 7, 1 };
List<int[]> asteroids = new() { new[] { 0, 3 }, …};
int pSpawn = 50;

while (true)
{
    MoveAsteroids();
    CleanUpAsteroids();
    SpawnAsteroid();
    MovePlayer();
    CollisionCheck();
    // -> if collision: break out of while loop
    Display();
}
```

Asteroiden Game

- Kann (muss aber nicht) mehr OOP verwenden im Game
- Aktuell: Game State aufgeteilt auf mehrere Variablen:

```
int[] player;
List<int[]> asteroids;
```

- Überlegung: gesamten **Game State in ein Objekt** verpacken

```
private class State
{
    public int[] player;
    public List<int[]> asteroids;

    public State(int[] _player, List<int[]> _asteroids)
    {
        player = _player;
        asteroids = _asteroids;
    }
}
```

```
public override Score Play(int leve
{
    State gameState = new State(
        new int[] { 7, 1 },
        new List<int[]>() { new[] {
    });
```

Hauptauftrag

- Mache **detailliertes Modell** für **dein Game** …
- … von Hand (Papier / OneNote / …)
- Programmieren verboten! Visual Studio bleibt geschlossen!
- **Modell:**
  - **a) Variablen**: Deklaration & Beschreibung
  - **b) Funktionen**: Beschreibung & *Model oder View*?
  - c) Inhalt **Play-Methode** skizzieren
- Mit LP besprechen

a)

Variablen:

- `int[] player;` // y & x Koordinate
- `List<int[]> asteroids;` // Liste
- …

b)

Funktionen:

- `MovePlayer();` // Überprüft User In
- `Display();` // zeigt Game an
- …

c)

```
int[] dim = { 8, 5 };
int[] player = { 7, 1 };
List<int[]> asteroids = new() { new[] { 0, 3 }, …};
int pSpawn = 50;


while (true)
{
    MoveAsteroids();
    CleanUpAsteroids();
    SpawnAsteroid();
    MovePlayer();
    CollisionCheck();
    // -> if collision: break out of while loop
    Display();
}
```