Dokumentation

Klassen:

- **BloomComponent**
- **BloomSettings**
  http://xbox.create.msdn.com/en-US/education/catalog/sample/bloom
  Die Klassen zur Implementierung des Bloom-Shaders, der zur Aufbesserung der Visuals dient. Unter Verwendung der Microsoft Permissive License (Ms-PL).

- **Bullets**

- **Button**
  Sofern die Maus mit dem Button überlappt, ändert sich jeden Frame der Farbwert des Buttons. So wird ein Blinken erzeugt. Buttonspezifische Größe hängt vom Viewport ab (Division durch dessen Breite und Höhe).

- **Camera**
  Die Kamera-Klasse beinhaltet eine Model-Matrix, die aus einer Skalierung und einer Translation besteht. Jene Matrix wird im SpriteBatch.Draw() Befehl verwendet und dient dazu, dass die Kamera der Spielfigur folgt. Dabei kann die Kamera aber die Map selbst nicht verlassen, da Links, Rechts, Oben und Unten anhand der Map-Größe abgegrenzt wird, die Kamera sich adaptiv anpasst, sofern die jeweiligen Ränder erreicht werden. Die scale-Variable steuert das Zoomen.

- **HighScore**
  Beinhaltet das Struct HighScoreData in dem die HighScore Daten gespeichert werden. Die Klasse selbst bietet zwei Methoden an, in Save/Load wird ein Pfad mitsamt dem HighScoreData struct als Argument übergeben. Ein Datenstrom wird geöffnet, der ein XML-Dokument liest oder erstellt und bei Bedarf die HighScoreDaten zurückgibt.

- **Map**
  Wird dazu verwendet, ein Array auszulesen, das den Indikator für die zu zeichnenden Tiles stellt. Die Methode Generate mit Array und Tilegröße als Parameter ist für dies zuständig.

- **MovingObject**
  Geschwindigkeit in Richtung X und Y wird bei Erstellung des Objekts übergeben. Kollision mit Tile führt zur Umkehr der Geschwindigkeit.

- **NameinputManager**
  http://www.dreamincode.net/forums/topic/158381-xna-text-input/
  Adaptiert von folgendem Link. Soll dazu dienen, den Namen im HighScore niederzuschreiben und fortan ausgeben zu lassen.

- **PixelPerfectCollision**
  An die Funktion IntersectsPixel werden 2 Rectangles sowie 2 Arrays gefüllt mit Farbwerten übergeben. Anhand einer Min/Max Berechnung wird der zu prüfende Bereich errechnet. Darauf folgend wird, insofern die Rectangles überschneiden, ein pixelbasierter Test durchgeführt. Überlappen zwei Pixel im Testbereich -> Kollision.

- **Player**
  Sofern der Player Beschleunigt, wird auf seine Position ein Vector2 gerechnet, der auf die derzeitige Rotation basiert. Dementsprechend müssen X und Y Achse separat voneinander auf Kollision geprüft werden. Dasselbe erfolgt, wenn der Spieler durch Reibung an Momentum verliert, sofern er nicht Beschleunigt. Zusätzlich wird in der Update Funktion des Players die Kollision mit feindlichen Objekten kontrolliert.

- **RectangleHelper**
  Statische Klasse, die dazu dient, festzustellen, ob der Player Oben, Unten, Links oder Rechts kollidiert.

- **SwitchandExit**
  Sofern Key(switch) eingesammelt, Teleportiert der Exit den Player in den nächsten Abschnitt.

- **Tiles**
  Wird dazu verwendet, die Tile-Texturen in das Map Objekt zu speichern.

- **Tower**
  Türme können Projektile abfeuern. Sie nehmen diese Aktion wahr, sofern sich der Spieler in einem bestimmten Radius befindet. Ebenfalls wird die Rotation des Turms so angeglichen, dass er in die Richtung des Spielers blickt, was über Math.Atan2(y, x) erfolgt. Projektile werden in eine Liste gespeichert, und verschwinden sofern sie eine gewisse Distanz vom Turm entfernt sind, oder mit einem Tile kollidieren. Der Turm kann jeweils nur alle zwei Sekunden schießen und maximal können nur zwei Projektile von einem Objekt, aufgrund eines Limits, bestehen.

- **Upgrade**
  Nicht implementiert.