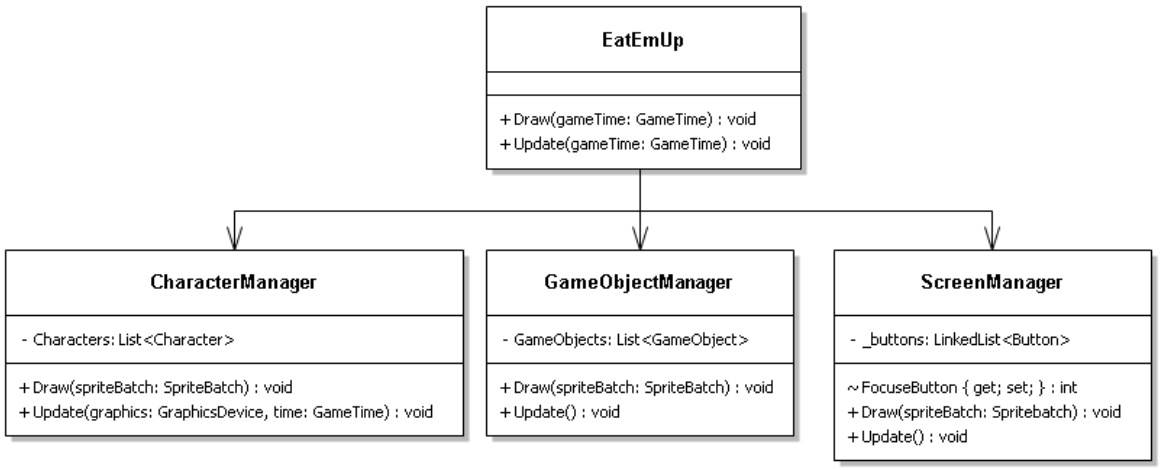
Eat 'Em Up – Technische Dokumentation

Inhalt

1. Organisationsklassen

1.1. EatEmUp

Diese Klasse ist abgeleitet von `Game` und ist somit die wichtigste Klasse. Sie steuert das zeichnen und aktualisieren der drei Organisationsklassen.



1.1.1. Wichtige Members und Properties

- `GameState CurrentGameState` bzw. `PreviousGameState` … momentaner bzw. voriger Status des Spiels (siehe `enum GameState`).
- `KeyboardState CurrentKeyboardState` bzw. `PreviousKeyboardState` … momentaner bzw. voriger Status der Tastatur. Werden benötigt um die Tastatureingaben der User zu verarbeiten.

1.1.2. Wichtige Methoden

- `void Draw(GameTime gameTime)` … je nach Spielstatus werden entweder die Draw-Methoden des CharacterManager und GameObjectManager oder des ScreenManager aufgerufen.
- `void Update(GameTime gameTime)` … ist zuständig für die Aktualisierung des Game- und KeyboardState und ruft die Update-Methoden der Manager-Klassen auf.
- `void UpdateGameState()` … aktualisiert den Spielstatus in Abhängigkeit des momentanen Spielstatus und des Tastaturinputs der User.

1.2. CharacterManager

Diese Klasse ist verantwortlich für die Initialisierung, Aktualisierung und Darstellung der beiden Charaktere „Lion“ und „Gazelle“.

1.2.1. Wichtige Members

`List<Character> Characters` enthält die Charaktere. Durch diese Liste können beide gleichzeitig angesprochen werden (z.B. beim Updaten und Rendern).

1.2.2. Wichtige Methoden

- `void Reset(bool swapControls = false)` … setzt die Charaktere auf ihre initialen Werte zurück und tauscht deren Steuerungen, falls von den Usern gewünscht.
- `void Update(GraphicsDevice graphicsDevice, GameTime gameTime)` … überprüft, ob die User ihre Ziele erreicht haben, die Gazelle ein Blatt eingesammelt und der Löwe Schuhe aufgesammelt haben und ruft die Update-Methoden der Charaktere auf.
- `bool SomeoneHasWon()` bzw. `Winner WhoHasWon()` … Geben zurück ob bzw. wer das Spiel gewonnen hat.

1.3. GameObjectManager

Diese Klasse ist verantwortlich für die Initialisierung, Aktualisierung und Darstellung aller Plattformen, Büsche, Blätter und Schuhe, sowie des LeafCounters.

1.3.1. Wichtige Members

Um alle Objekte gleichzeitig ansprechen zu können (z.B. zum Rendern) gibt es die Liste `List<GameObject> GameObjects`.

Außerdem gibt es mehrere Member zur Darstellung des sogenannten „LeafCounter“. Dieser zeigt an, wie viele Blätter die Gazelle bereits gesammelt hat und wie viele sie insgesamt sammeln muss.

1.3.2. Wichtige Methoden

- `void Draw(SpriteBatch spriteBatch)` … zeichnet die GameObjects, sowie den „LeafCounter“
- `void UpdateLeafCounter()` … aktualisiert den angezeigten Text des LeafCounters.
- `void RepositionLeaves()` … platziert jedes Blatt zufällig an einer neuen Position
- `void GenerateShoes()` bzw. `internal void HideShoes()` … platziert die Schuhe an einer zufälligen Position auf einer Plattform, bzw. lässt die Schuhe verschwinden, nachdem der Löwe sie aufgesammelt hat.

1.4. ScreenManager

Diese Klasse ist verantwortlich für die Darstellung des Start- und Pausemenüs, sowie der Gewinneranzeige und der Credits.

1.4.1. Wichtige Members und Properties

- `LinkedList<Button> _buttons` … Liste aller Menü-Buttons die die User betätigen können
- `Button FocusedButton` … der Button der momentan fokussiert ist. Im Setter dieses Propertys wird der zuvor fokussierte Button darüber informiert seinen Fokus zu verlieren und dem neuen Button wird der Fokus übergeben.
- `bool GazelleIsLeft` … zeigt an, ob die Gazelle im Menü links oder rechts gezeichnet werden soll (abhängig von der aktuellen Steuerung).

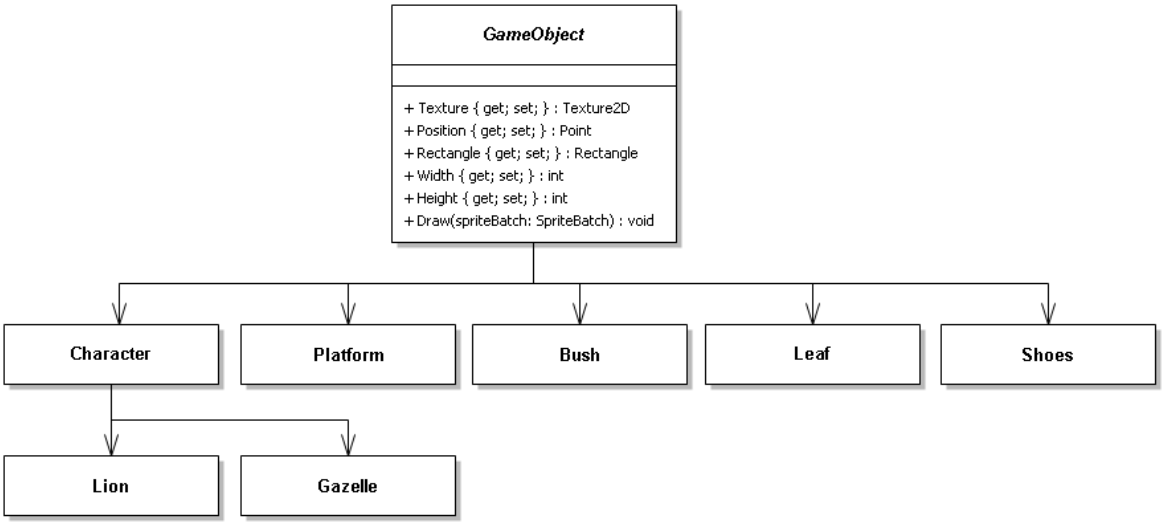
1.4.2. Wichtige Methoden

- `void Update()` … kümmert sich darum, welche Buttons angezeigt werden sollen (abhängig vom momentanen GameState).
- `void Draw(SpriteBatch spriteBatch)` … stellt die verschiedenen Menüs und zugehörigen Buttons, in Abhängigkeit vom aktuellen GameState, dar.
- `Button GetFocusedButton(Keys pressedKey = Keys.None)` … aktualisiert den Fokus der Buttons, wenn die User im Menü die nach oben oder nach unten Tasten gedrückt haben und gibt den momentan fokussierten Button zurück.

2. Darstellungsklassen

2.1. GameObject

`GameObject` ist eine abstrakte Klasse von denen jede Klasse abgeleitet ist, deren Objekte dargestellt werden und mit anderen Objekten interagieren können (Charaktere, Plattformen,…).



2.1.1. Members und Properties

- `Texture2D Texture` … Textur mit der das Objekt dargestellt wird
- `Point Position` … Position an der das Objekt dargestellt wird
- `Rectangle Rectangle` … Ausmaße des Objekts – diese Eigenschaft wird hauptsächlich für Kollisionsüberprüfungen benötigt
- `int Width` … Breite des Objekts
- `int Height` … Höhe des Objekts

2.1.2. Methoden

Die Klasse `GameObject` verlangt die Implementierung einer Methode `void Draw(SpriteBatch spriteBatch)` in der das Objekt gerendert wird.

2.2. Statische Spielelemente

Es gibt mehrere Objekte, die zwar dargestellt werden, sich aber nicht bewegen können und in keiner Weise Logik beinhalten. Diese Spielelemente werden vom GameObjectManager verwaltet.

- `class Platform : GameObject` … Darstellung der Plattformen. Diese Klasse beinhaltet einen zusätzlichen Member (`Rectangle _collisionRectangle`) da die Textur der Plattformen abgerundete Enden und Grasbüschel hat, welche nicht in die Collision Detection einfließen sollen (sonst würden die Spielfiguren nicht auf der Oberfläche der Plattform sondern ein paar Pixel darüber laufen, quasi auf der Spitze der Grashalme)
- `class Bush : GameObject` … Darstellung der Büsche.
- `class Leaf : GameObject` … Darstellung der Blätter.
- `class Shoes : GameObject` … Darstellung der Schuhe. Diese Klasse beinhaltet ein zusätzliches Property (`bool IsVisible`) da die Schuhe nicht dauerhaft gerendert werden sollen. Außerdem besitzt es die Methoden `void Show(Point position)` und `void Hide()` zum Anzeigen an einer bestimmten Position bzw. zum Verstecken der Schuhe.

2.3. Character

Die Klasse `class Character : GameObject` dient zur Darstellung der Spielfiguren.

2.3.1. Wichtige Members und Properties

- `Point _initialPosition` … Position an der die Spielfigur beim Beginn einer neuen Spielrunde startet.
- `Rectangle CatchingRectangle` … dieses Rechteck ist etwas kleiner als das Rechteck in dem die Spielfigur dargestellt wird. Es ist nötig, damit sich die beiden Charaktere für die User sichtlich „berühren“ müssen, dass der Löwe die Gazelle gefangen hat.
- `bool HasWon` … zeigt, ob dieser Charakter die aktuelle Spielrunde gewonnen hat.
- Diverse Members für die Bewegungen der Figur (springen, laufen, schnell laufen, verstecken).
- Diverse Members für die Steuerung der Figur.
- Diverse Members für die Animation der Figur.

2.3.2. Wichtige Methoden

- `void Update(GraphicsDevice graphics, GameTime gameTime)` … kümmert sich um die Bewegung des Charakters, die Animation und ob der Charakter schnell läuft bzw. sich hinter einem Busch versteckt.
- `void Draw(SpriteBatch spriteBatch)` … rendert die Figur, je nach Animation.
- `void Reset()` … wird aufgerufen, wenn eine neue Spielerunde beginnt. Sie setzt die Figur auf ihre Startposition zurück und löscht alle Statuswerte (gesammelte Blätter und Schuhe, erhöhte Geschwindigkeit,…).
- `void ResetControls(List<KeyValuePair<Controls, Keys>> controls)` … Überschreibt die Steuerung des Charakters mit den übergebenen Werten (wird benötigt, wenn die User die Rollen tauschen möchten).

- `void UpdatePosition(…)` … 2 Überladungen – eine behandelt die Standardbewegung der Figur, die andere behandelt die Bewegung, wenn sich der Charakter hinter einem Busch versteckt und der User dann nach links oder rechts drückt.
- `void AvoidPositionOutOfBounds(GraphicsDevice graphics)` … falls der Charakter am Rand des Spielfensters hinausläuft bzw. am unteren Rand hinausfällt, erscheint die Figur auf der gegenüberliegenden Seite wieder.
- Diverse andere Update-Methoden, die sich um die Bewegung des Charakters kümmern.
- `void UpdateAnimation(GameTime gameTime)` … kümmert sich um die Animation der Figur, abhängig von ihrer Bewegung und der vergangenen Zeit seit die Animation zuletzt aktualisiert wurde.

2.4. Lion und Gazelle

Diese beiden Klassen sind von `class Character` abgeleitet und werden für die Unterschiede von Löwe und Gazelle benötigt.

Zum Beispiel haben die beiden nicht die gleichen Gewinnbedingungen und bei einem Neustart des Spiels, müssen unterschiedliche Dinge zurückgesetzt werden. Außerdem geben sie unterschiedliche Geräusche von sich.

3. Utilities

3.1. Enums

3.1.1. GameState

Das `enum GameState` definiert die möglichen Zustände des Spiels:

- `Playing` … Zustand während die User gerade spielen
- `InitialMenu` … Startmenü
- `Pause` … Pausemenü
- `End` … Anzeige des Gewinners der letzten Spielrunde
- `Credits`

3.1.2. Enum Controls

Das `enum Controls` definiert die Steuerung der Charaktere. Es besteht aus folgenden Elementen:

- `Left`
- `Right`
- `Up`
- `Down`
- `Action`

3.1.3.Enum Winner

Das `enum Winner` wird für die Darstellung des richtigen Winner Screens benötigt und hat folgende Elemente:

- `Lion`
- `Gazelle`
- `None`

3.2. Klassen

3.2.1.StaticStrings

Die Klasse `static class StaticStrings` beinhaltet diverse statische Strings, die für das Laden von Inhalten benötigt werden.

3.2.2.KeyHelper

Die Klasse `static class KeyHelper` beinhaltet zwei Methoden:

- `bool IsKeyPressed(Keys key)`
- `bool IsKeyReleased(Keys key)`

Mithilfe dieser Methoden kann festgestellt werden ob eine Taste in diesem Moment gedrückt oder losgelassen wurde. Das wird benötigt, da man mit der Microsoft.XNA Klasse `KeyboardState` zwar feststellen kann, welche Tasten momentan gedrückt sind und welche nicht, jedoch gibt sie keine Auskunft über den genauen Zeitpunkt des Drückens oder Loslassens von Tasten.

3.2.3.RectangleHelper

Die Klasse `static class RectangleHelper` wird verschiedene Fälle der Collision Detection benötigt und beinhaltet folgende Methoden:

- `bool IsOnTopOf(this Rectangle r1, Rectangle r2)` … stellt fest, ob sich das erste Rechteck in einem bestimmten Bereich über dem zweiten Rechteck befindet.
- `bool IntersectsAnyObject(this Rectangle bounds, IEnumerable<GameObject> gameObjects)` … stellt fest, ob sich das Rechteck mit irgendeinem der übergebenen GameObjects überlappt.
- `bool IsInsideOf(this Rectangle r1, Rectangle r2)` … stellt fest, ob sich das erste Rechteck vollständig innerhalb des zweiten Rechtecks befindet.