Spieleprogrammierung in Unity

Vertiefungsarbeit zum Modul 154

Kevin Thalmann und Ursin Zimmermann

M154

Inhaltsverzeichnis

[2 Einleitung 2](#_Toc434831538)

[3 Bezug zur Einführung 2](#_Toc434831539)

[4 Projektvorgaben 2](#_Toc434831540)

[5 Informieren 3](#_Toc434831541)

[5.1 Unity 3](#_Toc434831542)

[5.1.1 Grundlegende Informationen 3](#_Toc434831543)

[5.1.2 Unity Ads 3](#_Toc434831544)

[5.1.3 Unity Analytics 3](#_Toc434831545)

[5.1.4 Asset Store 3](#_Toc434831546)

[5.1.5 Aufbau von Unity 4](#_Toc434831547)

[5.2 Visual Studio 5](#_Toc434831548)

[5.3 Versionierung 5](#_Toc434831549)

[6 Planung 6](#_Toc434831550)

# Einleitung

In dieser Vertiefungsarbeit werden wir mit Unity ein Spiel programmieren. Wir untersuchen dabei die Features und Services von Unity und werden dann eine Spiel-Idee ausarbeiten. Danach werden wir uns Gedanken über den Aufbau des Spieles aus Sicht der Programmierung machen.  
Bei der Implementation werden wir TDD verwenden und gleichzeitig Unit Tests in Unity ausprobieren. Am Schluss werden wir das Spiel Auf dem PC und / oder Handy veröffentlichen.   
Vorkenntnisse haben wir beide in C# und TDD jedoch keine in Unity.

# Bezug zur Einführung

In Unity werden wir vor allem die Implementation anschauen. Denn das einige was sich verändert ist die Entwicklungsumgebung. Das Design wird sich ein bisschen verändern, aber nicht viel da Unity auch Objekt Orientiert ist, jedoch wird alles auf Script aufgebaut, die dann zurechtgezogen werden.  
  
Wir werden auch TDD verwenden, was in Unity dank vorhandenem Framework einfach umzusetzen ist.

# Projektvorgaben

**Entwicklungsumgebung**: Unity, Visual Studio  
**Versionierung**: Github  
**Vorgehensweise**: TDD, IPERKA  
**Sprache**: Commits und Code in Englisch, Dokumentation n Deutsch.

# Informieren

## Unity

### Grundlegende Informationen

Auf der offiziellen Website von Unity werden 2 Versionen angeboten:

* Eine Kostenlose „Personal Edition“, welche weniger Features anbietet und bei jedem Start vom Spiel das Unity Logo anzeigt. Die Einschränkung hierbei ist, dass man ab 100‘000 Fr. Umsatz pro Jahr die „Professional Edition“ kaufen muss.
* Für 75$/Jahr oder einmalig 1500$ wird die „Professional Edition“ angeboten. Man erhält zusätzliche Features wie Cloud Build, oder besseren Support. Auch während der Programmierung erhält man mehr Features.

Links zum Verglich: <https://unity3d.com/get-unity>

Wir haben uns für Unity „Personal Edition entschieden“, da wir alle Features haben die wir brauchen und Ads und Analytics für beide Versionen zur Verfügung steht.

### Unity Ads

Unity Ads werden als kostenloser Service angeboten, bei dem man Werbung in sein Spiel schalten kann (nur bei Export zu Android und Apple) und damit Geld verdienen:  
<https://unity3d.com/services/ads>

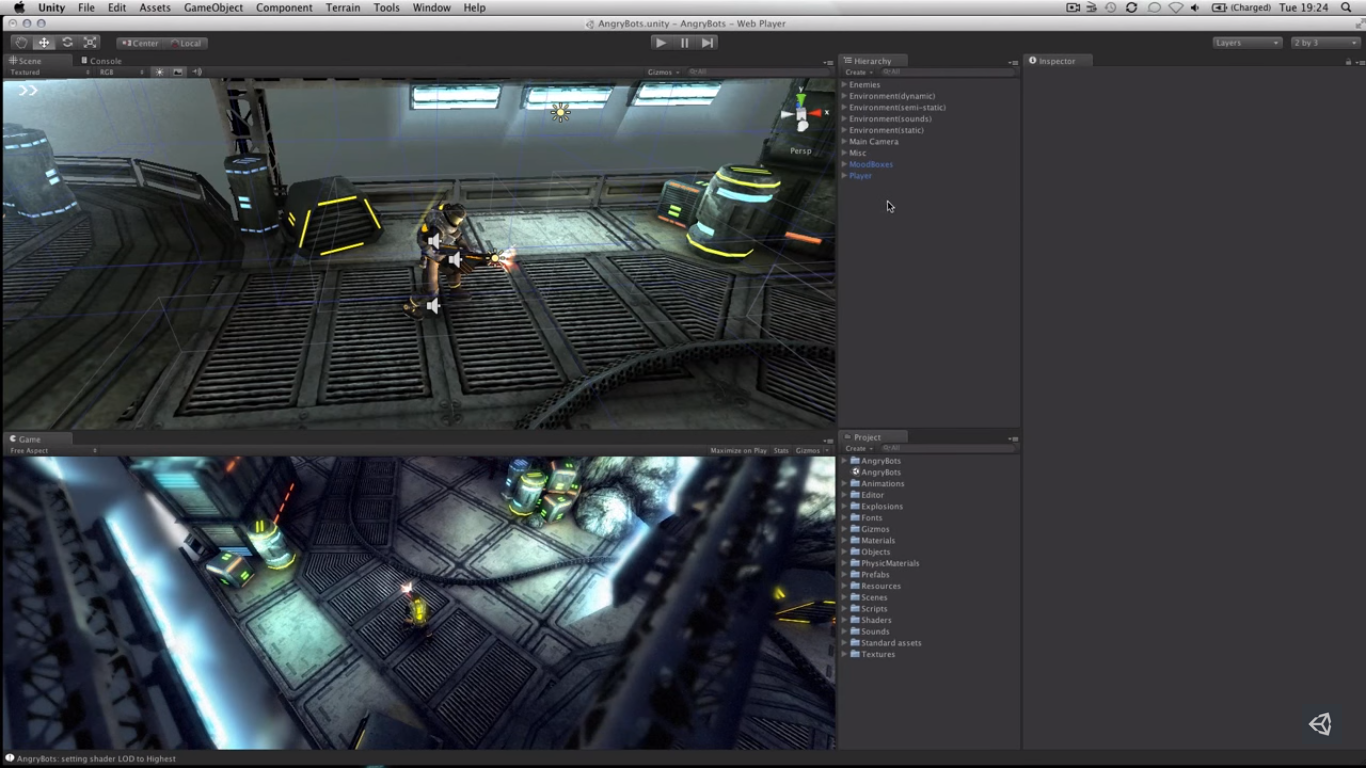
### Unity Analytics

Analytics werden von Unity auch kostenlos angeboten und können mit der „Personal Edition verwendet werden. Analytics erlauben es das Verhalten des Spielers und vieles mehr zu Analysieren und mit Grafiken darzustellen:  
<https://unity3d.com/services/analytics>

### Asset Store

Unity bietet einen Asset Store an auf dem man verschiede Figuren, Scripts und weiter herunterladen kann. Jedes Asset im Store unterliegt standardmässig den Unity Standard Lizenzen und können im Projekt verwendet und verändert werden, auch kommerziell. Die Assets werden von Unity selber oder anderen Community Mitgliedern erstellt:  
<https://unity3d.com/services/analytics>

### Aufbau von Unity

Unity besteht aus fünf verschiedenen Hauptbereichen. Diese sind:

* Scene View

Im Scene View sieht man die 3D oder2D Ansicht des aktuellen Standorts im Game den man bearbeitet.

* Game View

Im Game View sieht man wie das Projekt schlussendlich genau aussehen wird.

* Hierarchy

In der Hierarchy sieht man alle Objekte die im Projekt verwendet werden, wie zum Beispiel Texturen oder Objekte

* Project

Im Project Abschnitt ist die File Hierarchie aufgelistet mit allen Dateien des Projekts.

* Inspector

Im Inspector sieht man die Eigenschaften eines ausgewählten Objektes. Diese kann man auch direkt im Inspector bearbeiten.

## Visual Studio

Als IDE haben wir Visual Studio verwendet. Wir haben beide unsere Lizenz von DreamSpark genutzt. Unity hat seit der letzten Version den Support für Visual Studio verstärkt, was es vereinfacht Unity Code zu debuggen und in Unity einzubinden.

Unity und Visual Studio Support: <http://docs.unity3d.com/Manual/VisualStudioIntegration.html>

## Versionisierung

Zur Versionierung haben wir GitHub verwendet. Das Projekt ist dann zwar Open Source, aber wir können auch dann nach dem Projekt immer noch daran Arbeiten und müssen keinen Server haben um Git zu verwenden.  
 Wir haben uns geeinigt die Commits möglichst klein zu halten und nach Jedem Feature einen Commit zu machen. Wir machen dabei keine Branches für Features, das das Projekt zu klein ist.

Link zum Branch: <https://github.com/Saphatonic/M154>

# Planung

## Game Design

Die Spiel Idee ist ziemlich eingeschränkt, da wir nicht so viel Zeit haben und daher keine komplexen Spiele erstellen können.

Auch bietet der Unity Asset Store eine grosse Auswahl an freien Grafiken und Sounds an. Die selbigen können dann für unser Projekt verwendet werden. Deshalb müssen wir uns zuerst umschauen was es für Assets gibt, die wir verwenden können und dann eine Spiel Idee daran anpassen.   
Es können auch Grafiken und Sounds aus dem Internet verwendet werden, allerdings muss darauf geachtet werden, dass sie auch kommerziell verwendbar sind, falls wir das Spiel veröffentlichen wollen.

## Spielidee

Nachdem wir uns im Asset Store von Unity umgesehen haben, haben wir ein „Flappy Birds“ Asset Pack entdeckt, welches schon vorgefertigte Sounds und Grafiken hat. Wir dachten es wäre eine gute Übung diese Grafiken zu nehmen und ein „Flappy Birds“ Klon zu erstellen, da wir kein innovatives Spiel erstellen wollen, sondern nur Unity kennen lernen.

## Spielprinzip

Unser Spiel hiesst: „Tap and Fly“.  
Es geht darum wiederholt auf den Bildschirm zu tippen. Dabei fliegt der Vogel wiederholt hoch wenn man ihn antippt. Danach fällt er runter und der Spieler muss ihn erneut antippen um wieder hoch zu fliegen. Der Vogel fliegt dabei nach rechts und man muss Hindernissen ausweichen und bekommt dafür Punkte. Am Ende wird dann ein Highscore angezeigt.

## Grafik und Sound

Die Grafiken und Sounds werden von dem Asset verwendet, das wir im Unity Store gefunden haben.

<https://www.assetstore.unity3d.com/en/#!/content/21454>

## Monetarisierung

Wir blenden Werbebanner ein.

## Plattform

Das Spiel wird im Google Play Store herausgebracht, da einer von uns (Kevin) einen account hat und wir so das Spiel monetarisieren können.

# Umsetzung

## Assets downloaden

Die Assets konnten wir direkt in Unity herunterladen und dann importieren. Wir haben „Tap and Fly“ und die Unit Test Assets heruntergeladen. Danach konnten sie problemlos importiert werden und konnten im Projekt verwendet werden.

## Programmierung

Das Spezielle an Unity war, dass es bereits einen Game Loop gab, das Grundgerüst des Programms war also schon vorhanden. Wir mussten dann nur noch unsere Grafiken hinzufügen und ihnen „Komponenten“ zuweisen.

Komponenten sind auch das was Unity speziell machen. Der Sinn dabei ist, das Jedes Objekt aus verschiedenen Komponenten besteht, wie zum Beispiel der Grafik, der Gamelogik und einigen Modifiers die von Unity bereitgestellt werden um Berechnungen zu machen, wie Collision Detection, etc. So wird zum Beispiel der Charakter in die Szene gezogen, danach wird ihm ein Script hinzugefügt damit er laufen kann. Damit er nicht durch den Boden fällt wird ihm Collision Detection hinzugefügt.

So wird das ganze in Script umgesetzt und zurechtgezogen. Da wir noch Unit Tests verwendet haben war dies auch recht einfach, da alles Modular aufgebaut war und so einfacher zum Testen war.

### Unit Test

Wir haben beide schon mit Unit Tests gearbeitet, weshalb wir schon damit vertraut waren. In Unity waren die Unity Tests auch nicht anders. Es gab zusammen mit den Test Tools einen TestRunner mit dem wir immer wieder die Tests laufen lassen konnten und so sehen ob alles noch funktioniert.

Das Prinzip war hier, die tests zu erstellen und danach die Klasse. Danach die Klasse so anzupassen bis die Tests stimmen.

### Programm