TOWER DEFENSE PROJECT

Vladimir Malikov

David Asmuth

Steffen Windrath

Johannes Martens

Abschlussbericht

Tower Defense Project

# Spieldesign und Ablauf Seite 2 - 6

#### Spielfeld und seine Komponenten

1. Das Spielfeld
2. Die Bits
3. Die Leiterbahnen (Bit Trace)
4. Die Türme (Tower)
5. Der Prozessor (CPU)

#### Interface

1. Interface
2. Steuerung

#### Spielablauf

1. Startphase
2. Spielphase
3. Endphase

# Anleitung Seite 7 - 8

# Coding Styleguide und Projektstandards Seite 8-15

1. Namenskonventionen
2. Ordnungen

#### Unity und Projekt Anpassungen

1. C# Code Anpassungen
2. Struktur
3. Gültigkeit

# Systeme Seite 16-17

# Anwendungsfälle Seite 18-30

## Spielfeld und seine Komponenten

## Das Spielfeld

Das Spielfeld (siehe Abb.1) besteht aus 4 wichtigen Komponenten. Den Bits, die dem Spieler entweder als Gegner oder Ressource dienen. Den Pfaden die den Bits als Laufweg dienen. Den Türmen die Bits angreifen und dem Prozessor (CPU), der das Schutzobjekt darstellt.

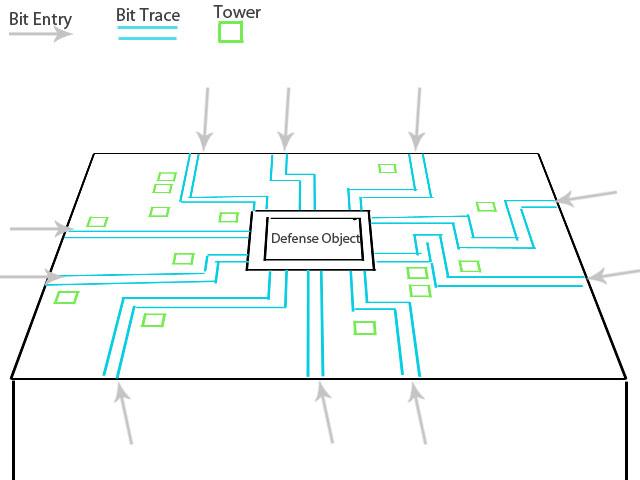


Abb.1 Schematische Darstellung des Spielfelds.

## Die Bits

Bits können drei Basiszustände haben (siehe Abb. 2).

Der Startzustand eines Bits ist der unbekannte Zustand. Ist das Bit unbekannt und erreicht den Prozessor, so wird dessen Abkühlrate kurzzeitig auf 0 gesetzt. Das heisst, die Wärme kann in diesem Zeitfenster nurnoch steigen. Das Bit kann mittels speziellem Turm identifiziert werden. Dadurch erhält es einen der beiden anderen Zustände:

Ist das Bit gutartig, bekommt der Spieler bei erreichen des Prozessors CPU-Zeit gut geschrieben.

Ist es Bösartig, erhitzt es den Prozessor beim erreichen. Türme versuchen diese auf dem Weg zu zerstören. Schaffen sie dies, erhält der Spieler Punkte.

Es gibt jeweils nocheinmal verstärkte Versionen des guten und des bösen Bits (siehe Abb. 2 “Advanced”), welche einen stärkeren Effekt auf die CPU haben. Zudem lässt sich die verstärkte Version des bösartigen Bits schwerer zerstören.

Dabei bewegen sich die Bits von der äußeren Kante des Spielfelds in die Mitte zu dem Prozessor (siehe Abb.1). Es gibt vordefinierte Punkte (“Bit Entry”) an denen die Bits erscheinen können. Sobald sie erscheinen, folgen sie einen festgelegten Weg (“Bit Trace”) und laufen ihn mit gleich bleibender Geschwindigkeit ab (sofern keine äußeren Einflüsse vorhanden).

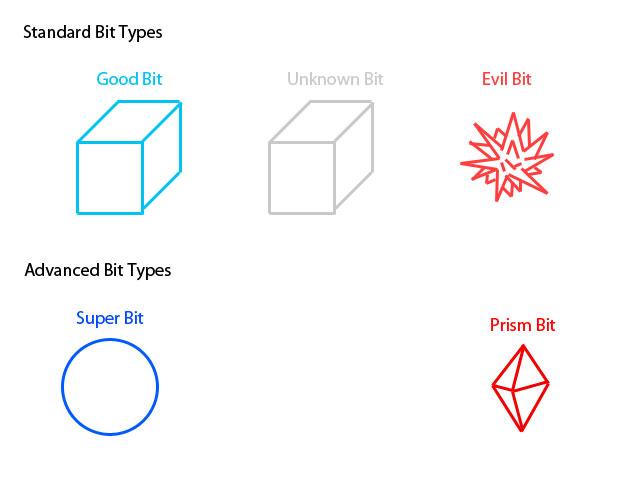


Abb. 2 die verschiedenen Bit-Typen

## 

## Die Leiterbahnen (“Bit Trace”)

Eine Leiterbahn (siehe Abb.1) ist das sichtbare Wegpunkt-System dem die Bits folgen. Leiterbahnen können an- und abgeschaltet sein. Zum Beginn sind die meisten Leiterbahnen inaktiv, werden jedoch im weiteren Spielverlauf zugeschaltet um den Schwierigkeitsgrad langsam zu erhöhen.

## 

## Die Türme (Tower)

Die Türme sind der elementare Bestandteil der Spiels. Sie bekämpfen oder manipulieren bösartige Bits, unterstützen andere Türme oder den Prozessor.

Türme muss der Spieler selbst bauen. Dies verbraucht Ressourcen (CPU-Zeit). Diese Ressourcen stellt der Prozessor bereit. Der Spieler kann die Türme frei Platzieren, jedoch nicht auf Leiterbahnen, oder der CPU.

Es gibt verschiede Arten von Türmen, diese unterscheiden sich in dem Preis ihrer Anschaffung und ihrer Attribute (Reichweite, Schussfrequenz, Schaden, Effekt). Diese sind der TurmArten.PDF zu entnehmen. Einen sonderfall nimmt der Detector-Tower ein, der lediglich unbekannte Bits in gut oder schlecht einteilt.

## Der Prozessor (CPU)

Ist der Zentrale Punkt des Spielfeldes. Die CPU hat zwei Ressourcen, Hitze und CPU-Zeit.

### Hitze

Die CPU hat eine standard Betriebswärme von Hmin und eine maximale Arbeitstemperatur von Hmax. Erreicht ein bösartiges Bit die CPU so erhöht sich die aktuelle Temperatur, und sinkt langsam wieder auf Hmin ab. Erreichen zuviele bösartige Bits die CPU innerhalb kurzer Zeit wird Hmax erreicht. Dies führt zur Zerstörung der CPU - das Spiel ist beendet.

### CPU-Zeit

Ist das “Geld” des Spielers. Er erhält CPU-Zeit wenn gutartige Bits die CPU erreichen, und verliert CPU-Zeit sobald er einen Turm erstellt. CPU-Zeit hat hier nichts mit Zeit im eigentlichen Sinne zutun. (D.h die Zeit vergeht nicht bei nicht Verwendung).

# Interface

Das Interface lässt sich in 2 Teile aufteilen. Zum Einen die Anzeige für die CPU (Temperatur und CPU-Zeit) und Punkte zum Anderen das Bau-Menü für die Türme. Beide Teile sind als 2D-Interface am unteren Rand des Bildschirms angebracht (Abb. 3).

Fährt der Spieler mit seiner Maus über einen Turm so erhält er mehr infos. Es kann durch mehrere Türme gescollt werden. Ist er zu “teuer” ist er ausgegraut. Mit einem Klick kann man ihn Platzieren.

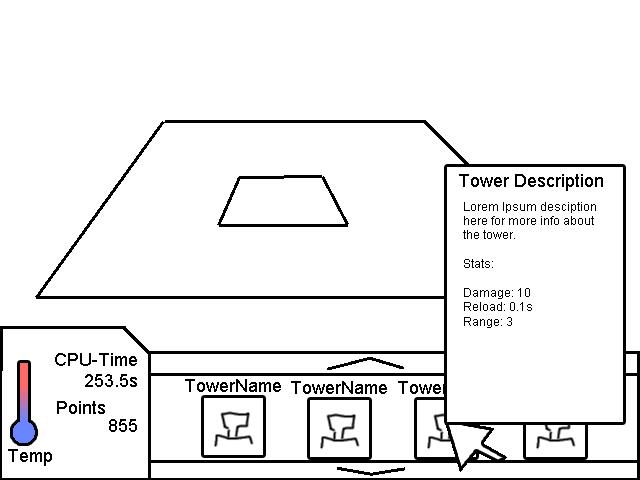


Abb 3. Schema der Spiel-Interface

## Steuerung

Das Spiel wird nur mit der Maus gesteuert. Bewegt man die Maus an einen Bildschirmrand bewegt man sich in diese Richtung. Scrollt man mit dem Mausrad, Bewegt man sich vom Spielfeld weg oder kommt diesem näher (Strategiespiel-typisch). Die Interaktion mit dem Interface erfolgt ebenfalls mit der Maus.

# Spielablauf

## Startphase

Der Spieler beginnt auf dem Spielfeld mit einer aktiven Leiterbahn und der CPU auf Betriebswärme.

Er hat gerade so viel CPU-Zeit um sich einen Detector-Tower zu erstellen (siehe Spielkomponenten->Türme ). Unbekannte Bits wandern in Zeitlichen Abständen über die Leiterbahn.

In dieser Situation ist der Spielablauf in einem Leerlaufmodus. Die CPU Temperatur bleibt auf Betriebswärme die unbekannten Bits sind in dem Fall ohne Effekt, der Spieler bekommt keine Punkte und CPU-Zeit. Dies ist wichtig um einen neuen Spieler Zeit zu geben sich mit der Umgebung vertraut zu machen.

Sobald der Spieler einen Detector-Tower gesetzt hat, fängt dieser an einige unbekannte Bits aufzudecken. Heraus kommen dann gute und böse Bits. Zunächst wird die CPU-Temperatur steigen, da der Spieler keine Türme besitzt um die bösen Bits zu eliminieren. Einem neuen Spieler wird so nebenbei verdeutlicht was passiert wenn ein böses Bit die CPU erreicht.

Bereits nach kurzer Zeit kann sich der Spieler den ersten Angriffs-Turm leisten und so seine CPU vor dem Überhitzen bewahren.

## Spielphase

Ab dieser Stelle entscheidet der Spieler frei den Spielverlauf; er kann sein Angriffs-Turm erweitern, einen weiteren Detector-Turm aufstellen um die unbekannten Bits los zu werden oder einen anderen Turm bauen.

Der Spieler muss nun entscheiden welche Türme an welchen Positionen taktisch sinnvoll sind um möglichst effizient zu bleiben.

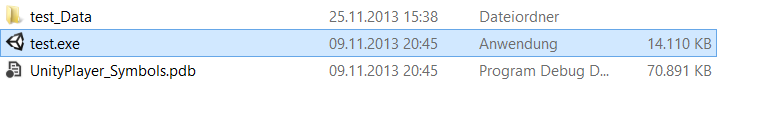
Nach bestimmten Punktständen werden weitere Leiterbahnen aktiv geschaltet und die Frequenz in der Bits erscheinen erhöht sich.

## Endphase

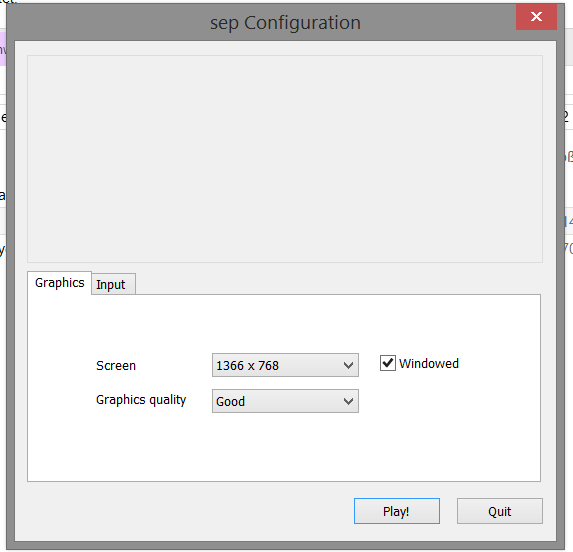
Je nach dem wie gut die Strategie des Spielers war wird er früher oder später den Ansturm der Bits nicht mehr bewältigen können. Die CPU überhitzt und die Spielrunde ist beendet. Der Spieler bekommt anschließend nocheinmal seine erreichten Punkte präsentiert.

**1. Fertigen Build starten.**

Stellen Sie sicher,dass der Ordner die Datei “<Name>.exe” und den Ordner “<Name>\_Data” enthält. Wenn eins davon fehlt, wird das Spiel falsch oder garnicht gestartet.



Wenn alles in Ordnung ist, könnt Ihr einfach durch Doppelklick auf <Name>.exe das Spiel starten. Es erscheint ein Einstellungsmenü:



Wählt die passende Einstellungen und klicken Sie auf “Play!”.

**2. Build in Unity3D starten**

**2.1 Update von Unity3D**

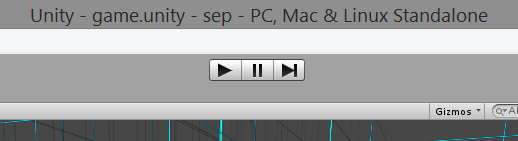
Laden sie die neueste Version(2.3) herunter. Und installieren Sie sie. Beim laden des Projekts(Siehe 2.2) kann eine Warnung angezeigt werden, dass das vorhandene Projekt zwar geöffnet werden kann, aber danach mit den älteren Versionen nicht mehr kompatibel ist. Klicken Sie auf “Ok”. Diese Fehlermeldung erscheint nicht, wenn ein Build geladen wird, der mit der aktuellen Version bereits bearbeitet wurde.

**2.2 Laden eines Builds in Unity3D**

Starten Sie Unity3D. Gehen Sie nun auf File->Open Scene. Wählen Sie im daraufhin erscheinenden Dialog die <Name>.unity Datei ihres Builds aus (z.B. tdp\sep\Assets\Scenes\game.unity). Anschließend wird der Build geladen und geöffnet.

**2.3 Starten eines Builds in Unity3D**

Klick auf “Play”-Knopf im Editor. (1. Knopf links).



Sind Fehler vorhanden, die den Start unmöglich machen, wird eine Warnung eingeblendet.

Sind Fehler vorhanden, die aber keinen Start verhindern, so werden die Konsole angezeigt.

## 

## C# Standards

Im folgenden sind Programmierrichtlinien aufgeführt die im offiziellen .NET Framework von Microsoft Anwendung finden.

### 1. Namenskonventionen:

* Klassen **und Methoden** werden im PascalCasing geschrieben. Also beginnend mit einem Großbuchstaben und jedes Teilwort erhält wieder einen großen Buchstaben.

Beispiel:

Richtig: ClientConnection

Falsch: clientConnection

* Methoden Argumente und lokale Variablen werden im camelCasing schrieben. Wie PascalCasing, jedoch ohne vorangehenden Buchstaben.

Beispiel:

Richtig: itemCount

Falsch: itemcount / ItemCount

* Keine ungarische Notation. Bedeutet: keine vorangehenden Typ-Identifizierer an den Variablen. Heutzutage zeigen einem IDEs via ToolTip den Typ und Lokalität an.

Beispiel:

Richtig: counter / name

Falsch: iCounter / strName / \_counter

* Keine SCHREIENDEN KONSTANTEN NAMEN. Vermeidet, auch wenn es Konstanten oder readonly Variablen sind, diese komplett Groß zu schreiben. Stattdessen ist hier PascalCasing zu verwenden. Der Grund ist das diese Schreibweise zu viel Aufmerksamkeit zieht, und somit den Lesefluss stört.

Beispiel:

Richtig: SuperEvilNpcBitName

Falsch: SUPEREVILNPCBITNAME

* Vermeidet Abkürzungen. Abkürzungen sind schlecht lesbar und können zu Inkonsistenz führen. In einem Dokument steht “towerHlp” im nächsten “towerHlpr” beide stehen für “towerHelper”. Einzige Ausnahme sind Abkürzungen die bereits im normalen (Informatiker-)Sprachgebrauch zu finden sind. (z.B. ID, XML, NPC, FTP usw.)

Beispiel:

Richtig: towerGroup

Falsch: towerGrp

* Natürliche Abkürzungen mit mehr als 2 Buchstaben werden im PascalCasing geschrieben. Grund: Erhöht die Lesbarkeit.

Beispiel:

Richtig: XmlParser / NpcController / TowerID

Falsch: XMLParser / NPCController

* Keine Unter\_Striche in Namen. Erhöht die Lesbarkeit. Wir sind hier nicht in Ruby.

Beispiel:

Richtig: TowerWeapon

Falsch: Tower\_Weapon

* Benutzung von vordefinierten Typ Namen anstatt System-Typ Namen. Im Gegensatz zu Java gibt es keine echten primitiven Daten-Typen. Alle Typen sind Objekte. Somit ist “string” und “String” dasselbe genau wie “int” und “Int32” oder “long” und “Int64”. Die vordefinierten Typen sind immer zu bevorzugen.

Beispiel:

Richtig: string name / double offset

Falsch: String name / Double offset

* Verwendung von “var” in lokalen Variablen. Ausnahme sind vordefinierte Datentypen (int, long usw.). Dies vermeidet Durcheinander insbesondere bei Generics.

Beispiel:

Richtig: var towers = new Dictionary<string, ConcurrentStack<Tower>>()

Vermeiden: Dictionary<string, ConcurrentStack<Tower>> towers = new Dictionary<string, ConcurrentStack<Tower>>();

* Klassennamen sind (zusammengesetzte) Nomen.

Beispiel:

Richtig: Shooter / DeadTower

Falsch: Shoot / Die

* Interfaces beginnen mit einem großen “I”. Da es kein “implements” und kein “extends” für Abhängigkeiten gibt, sondern nur den Doppelpunkt, weiss man sonst nicht ob dies nun ein Interface oder eine Vaterklasse ist.

Beispiel:

Richtig: class BasicTower : ITower, GameObject // Implementiert: ITower & Erweitert: GameObject

Falsch: class BasicTower : Tower, GameObject // Erweitert oder implementiert Tower?

* Der Klassenname ist gleich dem Dateinamen, sowie der Namespace ist gleich der Project-Ordnerstruktur

Beispiel:

namespace Unity.GameObjects.Towers // Pfad: Unity\GameObjects\Towers\

{

class BasicTower // Dateiname BasicTower.cs

{

}

}

* Enums werden im Singular geschrieben und ohne typen ausgezeichnet (Letzteres nur wenn man mit Bitfeldern arbeitet). Zudem sollte kein Suffix “Enum” an den Namen gehängt werden.

Beispeil:

Richtig: public enum Direction

{

North,

East,

South,

West,

}

Falsch: public enum DirectionsEnum : int

{

North = 1,

East = 2,

South = 3,

West = 5

}

### 2. Ordungen

* Geschwungene Klammern starten in einer neuen Zeile

Beispiel:

class BasicTower

{

}

* Statische Felder kommen vor Feldern, die vor Gettern und Settern kommen.

Beispiel:

class BasicTower

{

private static string TowerName;

private int number;

private double offset;

public Number { get { return number; } set { number = value; } }

// Konstruktoren hier

// Methoden hier

}

### 3. Zugriffe

* Zugriffe auf Felder anderer Klassen finden über Getter und Setter Methoden statt. Kein Direktzugriff.

Beispiel:

Richtig:

private int length;

public int Length { get { return length; } set { length = value;} }

Falsch:

public int Length;

## 

## Unity und Projekt Anpassungen

Da Unity den C#-Code schon während der Design-Zeit interpretiert (Serialisiert), um z.B. Öffentlich Felder im Unity-Editor anzeigen und bearbeiten zu können, müssen Anpassungen an den Standards vorgenommen werden.

### C# Code anpassungen

* Da der Unity-Editor nur öffentliche Felder serialisiert (also keine Getter/Setter), wir aber unsere Zugriffe von Außen sauber Kapseln wollen, müssen wir unsere privaten Felder als “SerializeField” markieren, damit diese dennoch von Unity serialisiert werden und im Unity-Editor erscheinen. Dies geschieht natürlich nur mit Feldern die wir auch im Editor benötigen.

Beispiel:

[SerializeField]

private float health;

* Namespaces für Komponenten entfallen, da Unity nicht damit arbeitet. Die OrdnerStruktur des Projects ergibt sich somit aus der Assit-Struktur in Unity, da der Programm-Code ebenfalls als Asset gewertet wird.
* Die geschwungene Klammerung beginnt NICHT in einer neuen Zeile. Da dies Platzsparender ist.
* Keine Variablen die nur aus einem Zeichen bestehen. Ausnahme sind Lauf-Variablen (z.B. in schleifen)
* Assets für die Benutzeroberfläche werden mit dem Prefix Gui gekennzeichnet.

Beispiel:

class GuiMenu

### Struktur

Anpassung an das Resource System.

* Die Asset-Struktur wird folgendermaßen zusammengesetzt: Da die C# Componenten immer einer bestimmten Gruppe Innerhalb des Projekts zuzuordnen sind (z.B. Komponente TowerWeapon zu der Gruppe der Türme) wird für diese Gruppe ein Asset-Verzeichnis angelegt. In diesem Verzeichnis liegen die C#-Komponenten. Alle weiteren sind sog. Resourcen, die vom Spiel ge- oder entladen werden können. Resourcen die zu dieser Gruppe gehören sind in einzelnen Sub-Verzeichnissen die dessen Typ beschreiben unterteilt.

Beispiel: ( [Verzeichnis] {File} )

[Assets] (Root)

|\_\_ [Tower]

|\_\_[Resources]

| |\_\_ [Models]

| | |\_\_ {Base.mdl}

| | |\_\_ {Cannon1.mdl}

| |\_\_ [Textures]

| | |\_\_ {BaseTex.tga}

| |\_\_ [Materials]

| | |\_\_ {BaseMat}

| |\_\_ [Sounds]

| |\_\_ {shoot.mp3}

| |\_\_ {reload.mp3}

|\_\_ {TowerAttack.cs}

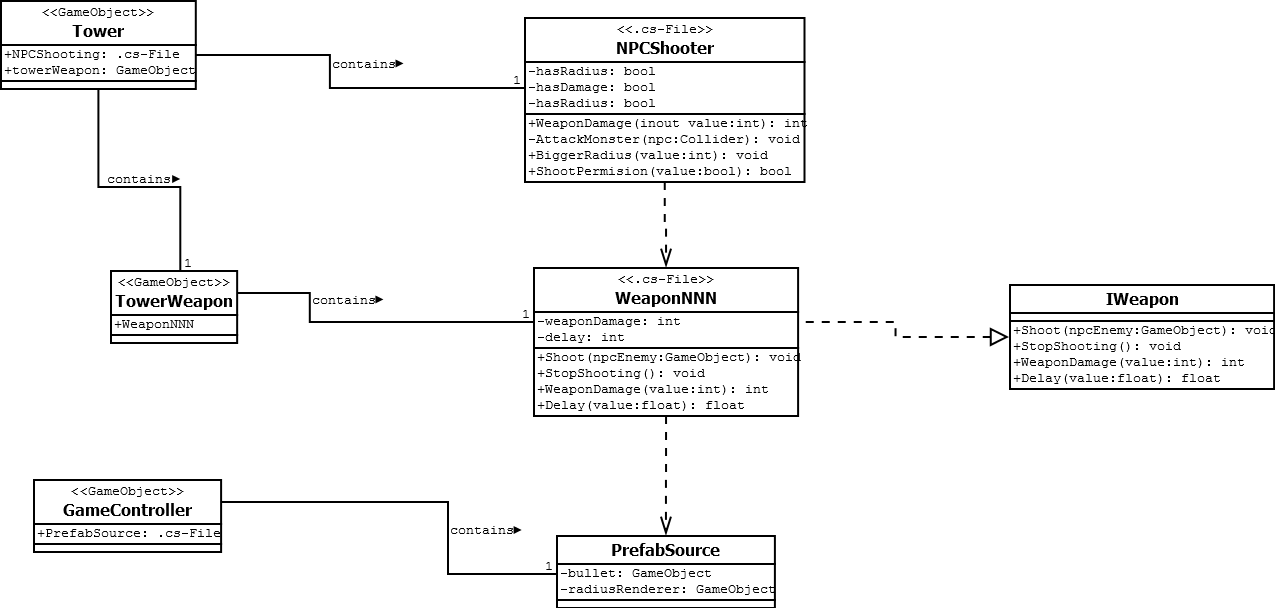
|\_\_ {TowerShootController.cs}

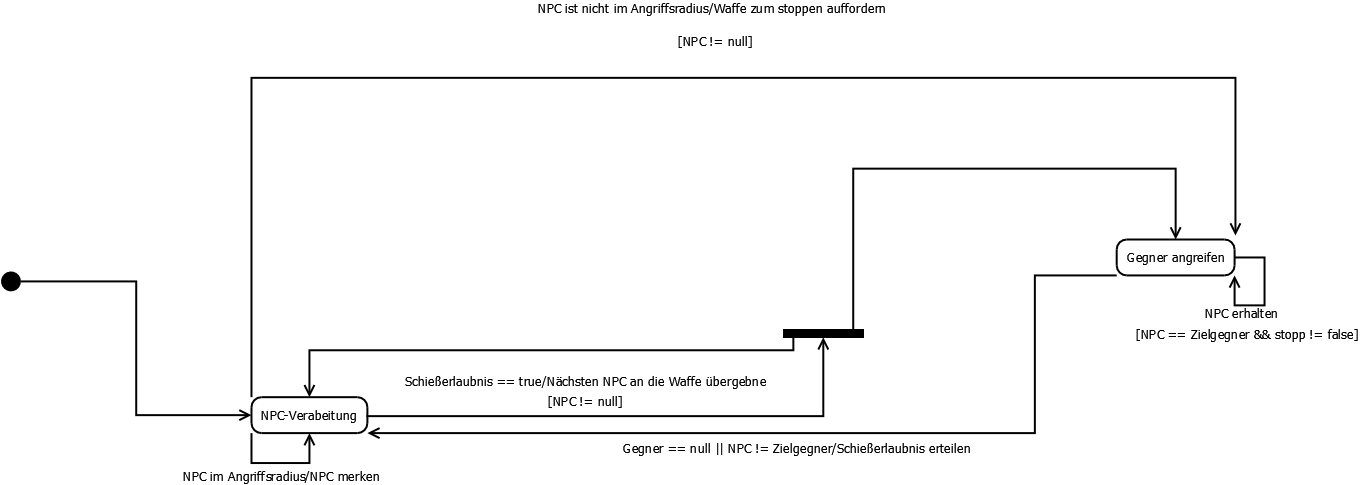
### Gültigkeit

Die Regeln sind spätestens beim Zuführen ins öffentliche Repository eingehalten.

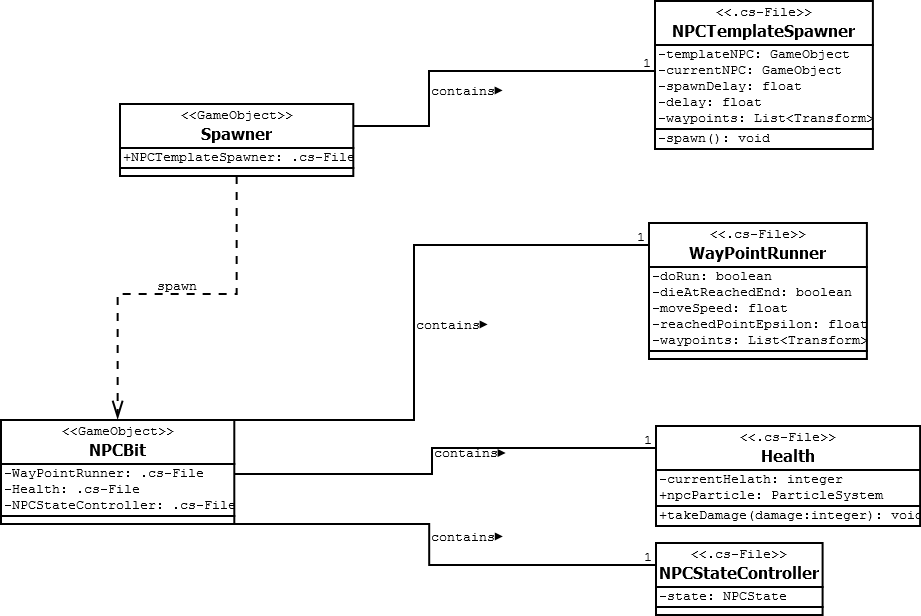
**Turm-System**

Die Türme sind wie folgt aufgebaut. Es gibt ein Sockel sowie die Turmwaffe selbst, was sich auf diesem Sockel befindet. Der Sockel ist der Kontrollpunkt für den Turm. Über den Sockel wird der Schuss ausgelöst, sowie die Turmwerte während des Spieles zu ändern. Somit stellt der Sockel ein Art Fassade dar. Das wird über .cs-Datei NPCShooter.cs realisiert. Diese Datei übernimmt aber auch ein Teil der Arbeit. Der Schuss wird in dieser Datei ausgelöst. Die Ausführung des Schusses wird in die WeaponNNN.cs delegiert. Wobei je nach Waffe steht NNN für den Waffentyp. Somit gibt es für jeden Waffentyp eine eigene WeaponNNN.cs-Datei. Um ein einheitlichen Zugriff zu ermöglichen, implementieren alle WeaponNNN.cs-Datei eine IWeapon-Interface.





**NPCBit-System**

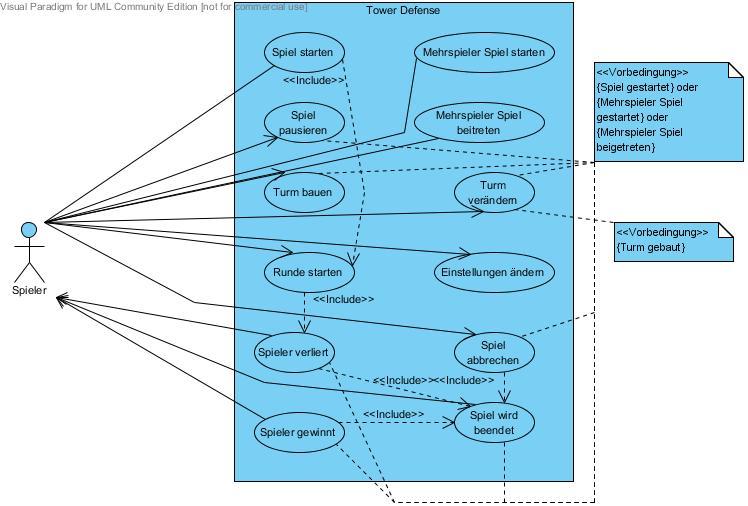
Die NPCBit sind die “Monster”. Die können entweder gutwillig oder böswillig sein. Der Zugriff auf NPCBit wird direkt ausgeführt. Um Leben zu ändern greift man auf Health.cs. Um Status zu ändern, greift man auf NPCStateController.cs

Der Spawner erstellt die neuen NPCBits, indem er sie klont. Danach die Wegpunkte übergibt und darauf wartet den nächsten zu machen.

**TD Projekt**

**9.06.2012**

**Ausarbeitenug der Anwendungsfälle**



**Name und Identifier:** Spiel starten

**Beschreibung :** Ein neues Spiel, bestehend aus mehreren Runden, wird gestartet. Wenn alle Runden beendet sind, wird das Spiel beendet.

**Beteiligte Akteure:** Spieler, Spiel

**Status:** bereit zum Review

**Verwendete Anwendungsfälle:** Runde starten

**Auslöser :**  Spieler hat im Hauptmenü den “Neues Spiel”- Button betätigt

**Vorbedingungen:** Noch kein Spiel gestartet

**Invarianten:** keine

**Nachbedingung / Ergebnis:**  Spieloberfläche /-Logik / etc. wird geladen

**Standardablauf :**  Es werden nacheinander alle Runden gestartet. Nach dem Ende der letzten Runde wird das Spiel beendet.

**Alternative Ablaufschritte:**

**Hinweise:**

**Änderungsgeschichte:**

**Name und Identifier:** Runde starten

**Beschreibung :** Die Runde startet wodurch die Gegner der Aktuellen Runde werden dem Spieler entgegengeschickt werden. Die Runde ist zu ende, wenn die Gegner der Runde entweder besiegt sind oder ihr Ziel erreicht haben.

**Beteiligte Akteure:** Spieler, Spiel

**Status:** bereit zum Review

**Verwendete Anwendungsfälle:** Spieler verliert.

**Auslöser :** Spieler hat im Spiel den “Starte Runde”- Button betätigt

**Vorbedingungen:** Spieler hat ein Spiel gestartet und es läuft momentan keine Runde

**Invarianten:** Spiel nicht abgestürzt

**Nachbedingung / Ergebnis:** Die Runde wird bis zu ihrem Ende abgearbeitet.

**Standardablauf :** Die Gegner werden dem Spieler mit bestimmten Abständen entgegengeschickt, bis alle für diese Runde vorgesehenen Gegner besiegt.

**Alternative Ablaufschritte:** Gegner erreichen ihr Ziel und der Spieler Verliert.

**Hinweise:** “Ziel” der Gegner ist es, der Burg des Spieler eine bestimmte Summe an Schaden zuzufügen oder dass eine bestimmte Anzahl Gegner eine Grenze überschreiten.

**Änderungsgeschichte:**

**Name und Identifier:** Mehrspieler Spiel starten

**Beschreibung :** Ein neues Spiel für mehrere Spieler wird gestartet. Wenn alle Runden des Spiels beendet sind, wird das Spiel beendet.

**Beteiligte Akteure:** Spieler, Spiel

**Status:** bereit zum Review

**Verwendete Anwendungsfälle:** Runde starten

**Auslöser :**  Spieler hat im Hauptmenü den “Neues Mehrspieler Spiel”- Button betätigt

**Vorbedingungen:** Noch kein Spiel gestartet

**Invarianten:** keine

**Nachbedingung / Ergebnis:**  Spieloberfläche /-Logik / etc. wird geladen

**Standardablauf :**  Bis zum starten der 1. Runde wird auf zusätzliche Spieler gewartet. Es werden nacheinander alle Runden gestartet. Nach dem Ende der letzten Runde wird das Spiel beendet.

**Alternative Ablaufschritte:**

**Hinweise:**

**Änderungsgeschichte:**

**Name und Identifier:** Mehrspieler Spiel beitreten

**Beschreibung :** Der Spieler tritt einem existierendem Mehrspieler Spiel bei. Wenn alle Runden des Spiels beendet sind, wird das Spiel beendet.

**Beteiligte Akteure:** Spieler, Spiel

**Status:** bereit zum Review

**Verwendete Anwendungsfälle:** Runde starten

**Auslöser :**  Spieler hat im Hauptmenü den “Mehrspieler Spiel beitreten”- Button betätigt

**Vorbedingungen:** Noch kein Spiel gestartet

**Invarianten:** keine

**Nachbedingung / Ergebnis:**  Spieloberfläche /-Logik / etc. wird geladen

**Standardablauf :**  Bis zum starten der 1. Runde wird auf zusätzliche Spieler gewartet. Es werden nacheinander alle Runden gestartet. Nach dem Ende der letzten Runde wird das Spiel beendet.

**Alternative Ablaufschritte:**

**Hinweise:**

**Änderungsgeschichte:**

**Name und Identifier:** Turm bauen

**Beschreibung :** Der Spieler baut im aktuellen Spiel einen Turm.

**Beteiligte Akteure:** Spieler, Spiel

**Status:** bereit zum Review

**Verwendete Anwendungsfälle:** keine

**Auslöser :**  Spieler hat auf ein bebaubares Feld geklickt

**Vorbedingungen:** Es wurde ein Spiel gestartet.

**Invarianten:** keine

**Nachbedingung / Ergebnis:** Ein bebaubares Feld wurde in einen Turm umgewandelt

**Standardablauf :**  Ein bebaubares Feld wird ausgewählt. Anschließend öffnet sich ein Menü, aus dem ein Turm ausgewählt werden kann. nachdem der Turm ausgewählt wurde schließt sich das Menü und das Feld wird durch den Turm ersetzt.

**Alternative Ablaufschritte:** Das Menü wird geschlossen, ohne dass ein Turm gebaut wurde.

**Hinweise:** Sollte der Spieler für einen oder mehr Türme nicht genügend Geld haben, können diese nicht ausgewählt werden.

**Änderungsgeschichte:**

**Name und Identifier:** Turm verändern

**Beschreibung :** Der Spieler baut im aktuellen Spiel einen Turm.

**Beteiligte Akteure:** Spieler, Spiel

**Status:** bereit zum Review

**Verwendete Anwendungsfälle:** keine

**Auslöser :**  Spieler hat auf ein bebaubares Feld geklickt

**Vorbedingungen:** Es wurde ein Spiel gestartet, es ist genügend Geld zum verändern des ausgewählten Turms vorhanden

**Invarianten:** keine

**Nachbedingung / Ergebnis:** Ein Turm wurde in einen anderen Turm oder ein bebaubares Feld umgewandelt.

**Standardablauf :**  Ein Turm wird ausgewählt. Anschließend öffnet sich ein Menü, aus dem ein Turm ausgewählt werden kann. nachdem der Turm ausgewählt wurde schließt sich das Menü und das Feld wird durch den Turm ersetzt.

**Alternative Ablaufschritte:** Statt einem Turm auszuwählen wird die Option gewählt, den aktuellen Turm abzureißen.

**Hinweise:**

**Änderungsgeschichte:**

**Name und Identifier:** Einstellungen ändern

**Beschreibung :** Das Spiel kann durch das Verändern von Einstellungen bedingt an die Wünsche des Spielers angepasst werden. Diese Einstellungen können z.B. die regulierung

**Beteiligte Akteure:** Spieler, Spiel

**Status:** bereit zum Review

**Verwendete Anwendungsfälle:** keine

**Auslöser :**  Spieler hat im Hauptmenü den “Einstellungen”- Button betätigt

**Vorbedingungen:** keine

**Invarianten:** keine

**Nachbedingung / Ergebnis:**  Einstellungen wurden verändert

**Standardablauf :** Es wird ein Fenster geöffnet, in dem verschiedene Möglichkeiten angeboten werden, die Einstellungen zu ändern, bis das Fenster geschlossen wird. Daraufhin ersetzten die neuen Einstellungen die alten.

**Alternative Ablaufschritte:**

**Hinweise:**

**Änderungsgeschichte:**

**Name und Identifier:** Spiel pausieren

**Beschreibung :** Der Ablauf des Spiels wird unterbrochen, bis der “Pause”-Button erneut gedrükt wird.

**Beteiligte Akteure:** Spieler, Spiel

**Status:** bereit zum Review

**Verwendete Anwendungsfälle:** keine

**Auslöser :**  Spieler hat im Spiel den “Pause”- Button betätigt

**Vorbedingungen:** Es wurde ein Spiel gestartet

**Invarianten:** keine

**Nachbedingung / Ergebnis:**  Das Spiel ist nach dem Ende der Pause unverändert.

**Standardablauf :** Der Ablauf des Spiels wird unterbrochen, bis der “Pause”-Button erneut gedrükt wird.

**Alternative Ablaufschritte:**

**Hinweise:**

**Änderungsgeschichte:**

**Name und Identifier:** Spiel wird beendet

**Beschreibung :** Das aktuelle Spiel wird beendet.

**Beteiligte Akteure:** Spieler, Spiel

**Status:** bereit zum Review

**Verwendete Anwendungsfälle:** keine

**Auslöser:**  Durch die Anwendungsfälle “Spieler gewinnt”, “Spieler verliert” oder “Spiel abbrechen” ausgelöst.

**Vorbedingungen:** Es wurde ein Spiel gestartet

**Invarianten:** keine

**Nachbedingung / Ergebnis:** Der Spieler befindet sich im Hauptmenü

**Standardablauf :** Das aktuelle Spiel wird beendet. das Hauptmenü wird aufgerufen.

**Alternative Ablaufschritte:**

**Hinweise:**

**Änderungsgeschichte:**

**Name und Identifier:** Spiel abbrechen

**Beschreibung :** Der Spieler beendet das aktuelle Spiel ohne zu gewinnen oder zu verlieren

**Beteiligte Akteure:** Spieler, Spiel

**Status:** bereit zum Review

**Verwendete Anwendungsfälle:** Spiel wird beendet

**Auslöser:**  Spieler hat im Spiel den “Spiel beenden”- Button betätigt

**Vorbedingungen:** Es wurde ein Spiel gestartet

**Invarianten:** keine

**Nachbedingung / Ergebnis:** Der Spieler befindet sich im Hauptmenü

**Standardablauf :** Der Anwendungsfall “Spiel wird beendet” wird ausgelöst

**Alternative Ablaufschritte:**

**Hinweise:**

**Änderungsgeschichte:**

**Name und Identifier:** Spieler verliert

**Beschreibung :** Der Spieler verliert und das Spiel wird beendet

**Beteiligte Akteure:** Spieler, Spiel

**Status:** bereit zum Review

**Verwendete Anwendungsfälle:** Spiel wird beendet

**Auslöser:**  Die Gegner haben ihr Ziel erreicht

**Vorbedingungen:** Es wurde ein Spiel gestartet

**Invarianten:** keine

**Nachbedingung / Ergebnis:** Der Spieler befindet sich im Hauptmenü

**Standardablauf :** Spieler erhält eine Meldung, dass er das Spiel verloren hat. Der Anwendungsfall “Spiel wird beendet” wird ausgelöst

**Alternative Ablaufschritte:**

**Hinweise:** “Ziel” der Gegner ist es, der Burg des Spieler eine bestimmte Summe an Schaden zuzufügen oder dass eine bestimmte Anzahl Gegner eine Grenze überschreiten.

**Änderungsgeschichte:**

**Name und Identifier:** Spieler verliert

**Beschreibung :** Der Spieler gewinnt und das Spiel wird beendet

**Beteiligte Akteure:** Spieler, Spiel

**Status:** bereit zum Review

**Verwendete Anwendungsfälle:** Spiel wird beendet

**Auslöser:**  Der Spieler hat alle Runden beendet, ohne dass die Gegner ihr Ziel erreicht haben

**Vorbedingungen:** Es wurde ein Spiel gestartet

**Invarianten:** keine

**Nachbedingung / Ergebnis:** Der Spieler befindet sich im Hauptmenü

**Standardablauf :** Spieler erhält eine Meldung, dass er das Spiel gewonnen hat. Der Anwendungsfall “Spiel wird beendet” wird ausgelöst

**Alternative Ablaufschritte:**

**Hinweise:** “Ziel” der Gegner ist es, der Burg des Spieler eine bestimmte Summe an Schaden zuzufügen oder dass eine bestimmte Anzahl Gegner eine Grenze überschreiten.

**Änderungsgeschichte:**