Serious Games SS2020 Übung 03

2D Project: Pooling and Multiple weapons
Abgabe: 19.05.2020, 10:00

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Abgabetermin: 19.05.2020, 10:00

Allgemeine Informationen

- Die Übungen können in Teams von bis zu 4 Personen bearbeitet werden.Die Abgabe erfolgt im Informatik-Moodle (*https://moodle.informatik.tu-darmstadt.de/course/view.php?id=831*) in einem Zip-Archiv (Upload durch ein beliebiges Gruppenmitglied).
  Das komplette Unity-Projekt ist abzugeben. Testen Sie vor der Abgabe, ob das Projekt mit nur den im Archiv vorhandenen Dateien geladen werden kann.
- Der abgegebene **[UPDATE]** *Projekt*ordner muss folgender Namenskonvention genügen:
  *Name_Vorname*_Uebung_*Übungsnummer*.
  Bspw. Mustermann_Max_Uebung_01 (So dass in Unityhub dieser Name angezeigt wird)
- Geben Sie den Theorieteil einer jeden Übung im .pdf Format ab, in der Sie **alle** Namen der Beteiligten auflisten, den Abgebenden markieren und die Übungsnummer angeben.
- Sie dürfen alternativ das Projekt in Form eines Links zu einem Gitrepository abgeben. Legen Sie dazu ein Release im Repository an und benennen Sie es *Uebung<XX>*. Der Zeitstempel des Repositories muss natürlich vor der regulären Abgabefrist sein, sonst wird die Übung als nicht bearbeitet gewertet.
  In Moodle geben Sie dann im .pdf Ihre Namen, Antworten zum Theorieteil und den Link zum Repository an.
  Achten Sie darauf, dass das Repository private ist. Laden Sie anschließend *SeriousGames-VL2020* als Collaborator ein.
- Namenskonvention in einer GitAbgabe: Benennen Sie auch hier den *Projekt*ordner nach der oben benannten Namenskonvention um.
- Der Abgabetermin steht immer auf dem jeweiligen Aufgabenblatt. Spätere Abgaben werden **nicht** berücksichtigt.
- Bei Fragen zu den Übungen verwenden Sie bitte das Moodle-Forum.
- Die zu verwendende Unity3D Version ist *2019.3.5f1*
- In jeder Übung können bis zu **10** Punkte erreicht werden.
- **Von den maximal zu erreichenden 10 Punkten wird für *jede* nicht eingehaltene, hier genannte Formalie *1* Punkt abgezogen!**

1 Unity3D Aufgaben[5]

1.1 Setup

Verwenden Sie Ihre Lösung der Letzten Übung, oder laden Sie sich die Musterlösung herunter und nutzen Sie diese.

1.2 ObjectPooling[1]

In Unity3d ist das Instantiieren und Zerstören von *GameObjects* recht teuer (bezogen auf Performance), deshalb nutzt man oft bei Objekten, die immer wiederkehren, eine Technik namens *ObjectPooling*. Grob erklärt: anstatt ein Objekt zu zerstören, deaktiviert man es nur, um sobald es wieder gebraucht wird, anstatt ein neues Objekt zu instantiieren, aktiviert man das vorherige Objekt wieder. Instantiiert wird nur noch, wenn kein inaktives Objekt gerade verfügbar ist.
Das trifft natürlich auf unsere *Projectiles* zu. Dementsprechend ist die erste Aufgabe einen *Objectpool* für unsere *Projectiles* zu implementieren.
Da wir unterschiedliche Projektile haben (Prefab Variants), brauchen wir aber auch mehrere *ObjectPools*, da wir ja sonst die falschen Projektile bekommen können, wenn wir alles in einen Pool werfen.

1.3 Weapon[1]

Derzeit schießt unser Schiff immer das *Projectile*, das der im Inspector gesetzte Pool ihm gibt. Nun möchten wir Waffen implementieren.
Waffen unterscheiden sich in Cooldown (Schussfrequenz) und Projektil.
Entwerfen Sie das Script *Weapon*. Ein *GameObject*, das dieses Script besitzt, wird dem Schiff als Kindobjekt zugewiesen.
Der PlayerController soll nun keinerlei Werte mehr besitzen, die sich um das Schießen drehen (Cooldown, ProjectilePool...).
All das gehört nun in das Weaponscript.
Der PlayerController kennt lediglich seine Weapon und bei Linksklick ruft er eine öffentliche Shoot() Methode auf.
Die Weapon kümmert sich dann um Cooldown, Projectile etc.

1.4 More Weapons[1]

Erstellen Sie neue *Waffen*. Es sollte nun einfach sein, neue *Waffen* hinzuzufügen, die neue *Projectiles* verschießen und deren *Schussfrequenz* sich unterscheidet.
Nun sollen Sie einen neuen *Waffen*typen erstellen, der statt 1 Schuss, eine gegebene Anzahl an Schüssen fächerartig gleichzeitig schießt. Hier könnten Sie bspw. mit Vererbungshierarchien arbeiten, damit die neue *Waffe* den *Collectables* und dem *Spieler* ohne Probleme zugewiesen werden kann ;).

1.5 Weapon Switching[1]

Es wäre sicherlich schön zwischen mehreren Waffen zu wechseln... Wenn der Spieler die rechte Maustaste betätigt, oder das Mausrad dreht (steht ihnen frei), soll er zwischen mehreren Waffen wechseln können.

1.6 Collectables[1]

Für diese Aufgabe benutzen wir zum ersten mal eine Art Kollision. Betrachten Sie Physik und Kollisionen erst einmal als Blackbox. Nächste Übung befassen wir uns mehr damit.
Ihr Spieler Benötigt die Components *Rigidbody2D* und *CircleCollider2D*. Stellen Sie im *Rigidbody2D* den Gravity Scale auf 0.
Nun erstellen Sie ein neues Objekt und ein Script *Collectable*. Dieses Objekt soll ebenfalls einen *CircleCollider2D* (keinen Rigidbody2D) erhalten. Stellen Sie hier *Is Trigger* auf true.

Im Script *Collectable* fügen Sie die Methode *void OnTriggerEnter2D(Collider2D collider)* hinzu. Diese Methode wird von einem *Monobehaviour* aufgerufen, wenn ein anderer *Collider* in seinen Bereich tritt. Wir möchten, dass wenn der Spieler also auf die Position des *Collectable* trifft (quasi kollidiert, aber ein Trigger ist keine Kollision), dass das *Collectable* überprüft, ob es der Spieler ist (Der Parameter *collider* gehört dem anderen Objekt ;)), und wenn ja, dem Spieler eine neue Waffe zuweist und sich danach selbst zerstört.

2 Theorie[5]

- Erklären Sie eine Anti-Cheat-Maßnahme in Mehrspielerspielen [2]
  Suppressed Update -> dead reckoning
- Welche Netzwerkarchitektur sollten Sie verwenden, wenn Sie ein Spiel mit Potential zum e-Sport entwickeln? Warum? [2]
  Client-Server. Beste Architektur zum Verhindern von Cheats
- Was ist der *Ping* im Kontext Netzwerke? [1]
  Die Zeit in ms, die die aktuelle Verbinsung benötigt, um eine Antwort vom Server zu erhalten