Projektdokumentation

des HS-ES-Reaktors

Inhalt

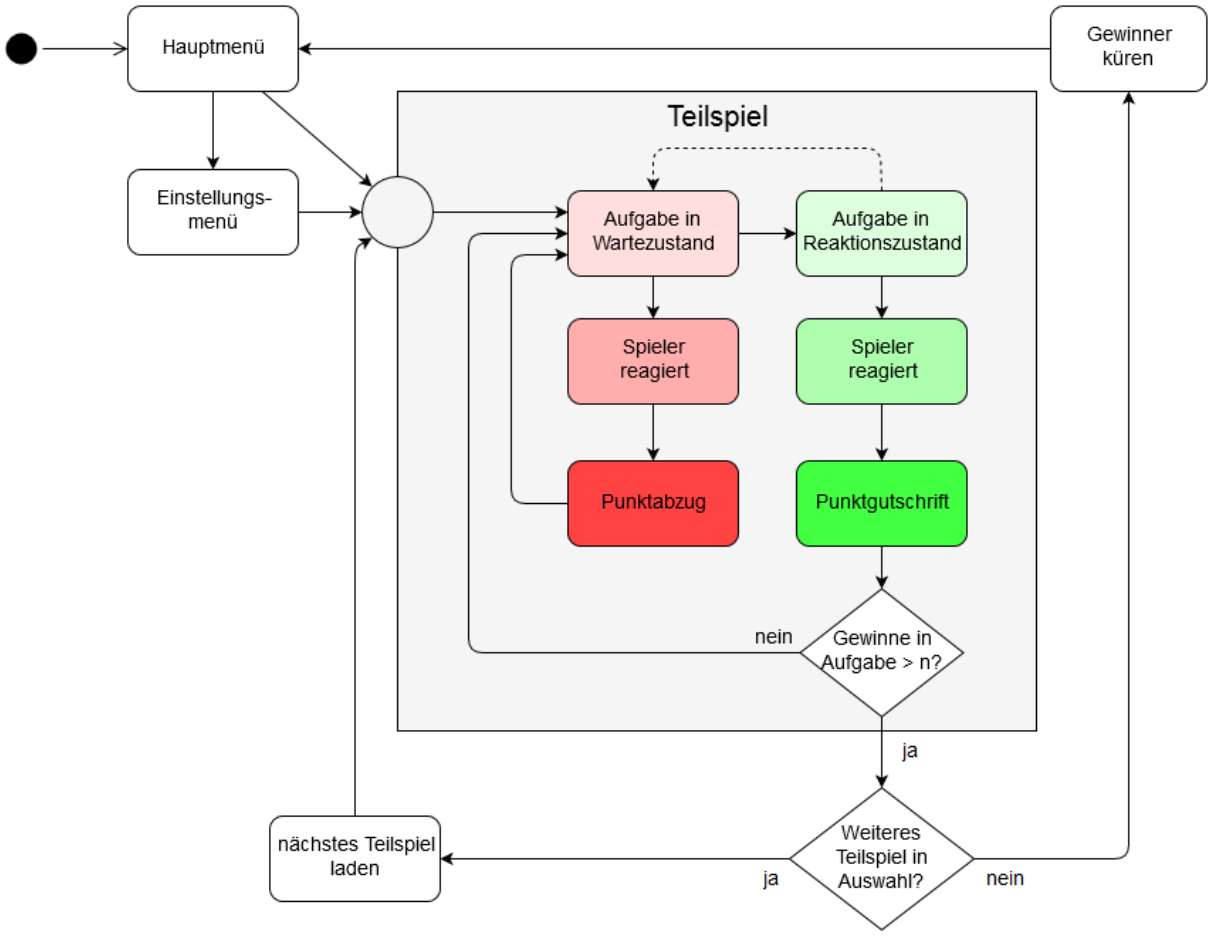
Tobias Deißler, Studiengang SWM
Hochschule Esslingen

Hochschule Esslingen
University of Applied Sciences

Spielbeschreibung

Der HS-ES-Reaktor (Hochschule-Esslingen-Reaktor) ist ein in Unity3D mithilfe von Visual C# umgesetztes Spiel für 1-4 Teilnehmer.
Ziel des Spiels ist es, in verschiedenen gestalteten Aufgaben sein Wissen über die Hochschule und seine Reaktionsschnelligkeit unter Beweis zu stellen und somit die meisten Punkte zu sammeln.

Verlauf des Spiels



Szenen

Szene „MenuMain“

Die „MenuMain“-Szene stellt das Hauptmenü dar, welches beim Anwendungsstart als erste Szene erscheint. Es dient zur groben Auswahl von Einstellungen und soll einen schnellen Spielstart ermöglichen. Weiterhin ist von dort aus ein Menü zu öffnen, von dem aus sich detailliertere Einstellungen treffen lassen.

Szene „MenuConfig“

Die „MenuConfig“-Szene dient als Menü für das Anwählen der im Spielverlauf gewünschten Teilspiele (*Subgames*). Teilspiele werden durch eine kurze Aufgabenbeschreibung repräsentiert. Befindet sich ein Haken neben der Aufgabenstellung, so wird dieses Teilspiel im Verlauf des Spiels aufgerufen. Die Kombination aus Sollten nicht alle Auswahlmöglichkeiten anzeigbar sein, so lässt sich die Liste vertikal scrollen. Aus der Szene heraus sind weiterhin der Start des angepassten Spiels, sowie die Rückkehr ins Hauptmenü unter Speichern der getroffenen Einstellungen möglich.

Szene „Subgame“

Die „Subgame“-Szene wird bei Spielstart aufgerufen, falls eine Auswahl an Teilspielen getroffen wurde. In ihr findet das eigentliche Spielgeschehen statt. Sie beinhält zur Laufzeit verschiedene Steuerungselemente: Für jeden Spieler wird ein Buzzer zur Aufnahme seiner Reaktion, sowie ein Textfeld zur Ausgabe von Meldungen durch das Spiel, wie z.B. die Beurteilung der Reaktion erzeugt. Diese sind beide Teil des „PlayerControls“-Prefabs. Im Zentrum der Szene wird zur Laufzeit das Prefab des nun folgende Teilspiels instantiiert und nach seinem Ende durch eine Instanz des nächsten Teilspiels ersetzt.

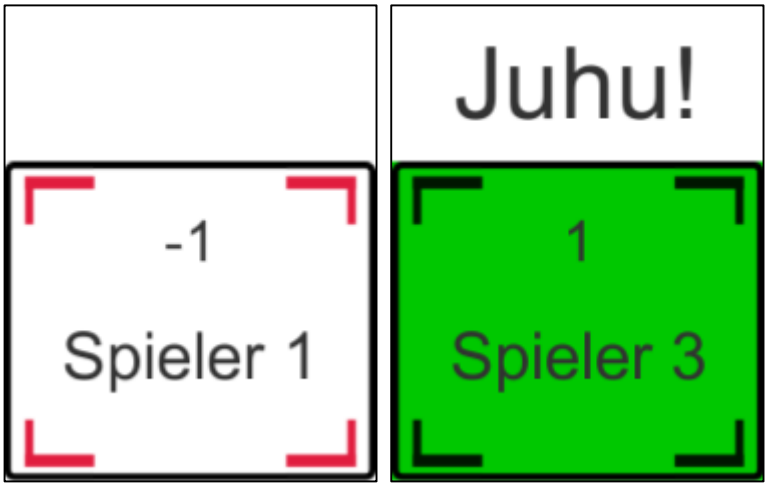
Szene „GameCompletion“

Die „GameCompletion“-Szene erscheint nach Ablauf des letzten Teilspiels. Sie gibt eine Zusammenfassung über das Ergebnis des Spiels durch eine Gratulation an den Sieger sowie das Auflisten der Teilnehmer zusammen mit ihrer Punktzahl, angefangen mit dem besten Ergebnis. Von dort können ein Neustart des Spiels mit den vorherigen Einstellungen und eine Rückkehr ins Hauptmenü erfolgen.

Ungebundene Prefabs

Ungebundene Prefabs, sind Prefabs, die vor dem Start der Anwendung in keiner Szene vorhanden sind. Diese werden erst zur Laufzeit instantiiert.
Zur einfacheren Betrachtung sind diese hier aufgelistet und kurz beschrieben.

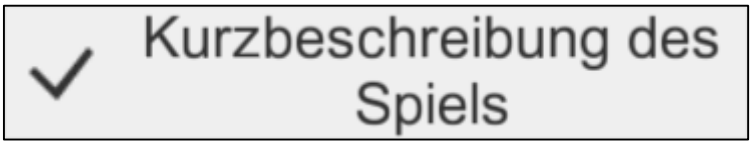
Prefab „PlayerControls“



Das PlayerControls-Prefab stellt eine grafische Schnittstelle zwischen dem Spiel und dem Teilnehmer zur Verfügung. Links ist das Aussehen im Standardzustand dargestellt. Rechts sieht man, dass sich bei korrektem Reagieren der Button grün färbt. Weiterhin können oberhalb des Buttons kurze Texte an den Spieler eingeblendet werden.

Buzzer	Button zur Reaktionsmeldung durch den Spieler
Buzzer/PlayerName	Text, der den Namen des Spielers beinhält
Buzzer/Score	Text, der die aktuelle Punktezahl des Spielers zeigt
Notification	Text, der eine Meldung an den Spieler beinhalten kann

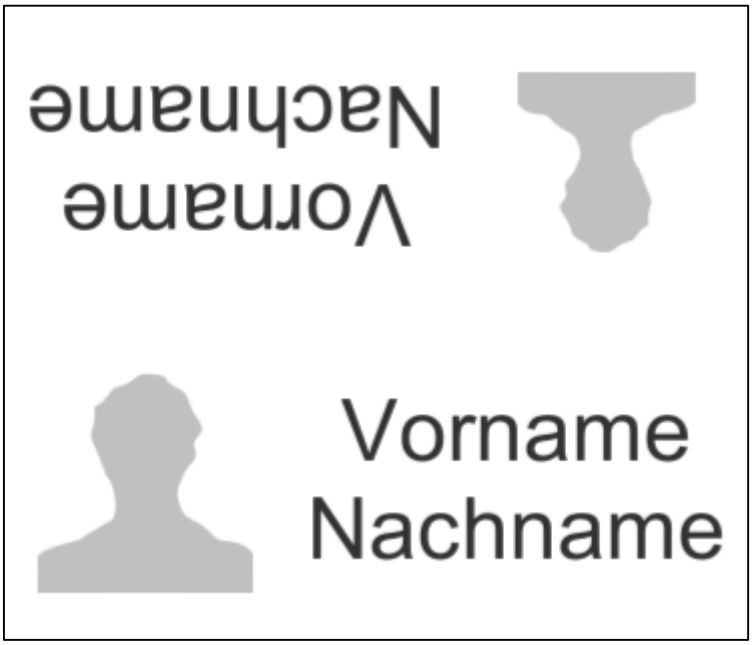
Prefab „SubgameToggle“



Das SubgameToggle-Prefab dient als Element, das eine kurze Beschreibung des Teilspiels enthält und mit einer rahmenlosen Checkbox anzeigt, ob das Spiel ausgewählt ist. Diese Checkbox lässt sich je nach Wunsch aktivieren oder deaktivieren.

Toggle	Interaktive Checkbox, mit oder ohne Haken
Toggle/Checkmark	Bild des Hakens der Checkbox
Text	Text, der eine Kurzbeschreibung des Teilspiels enthält

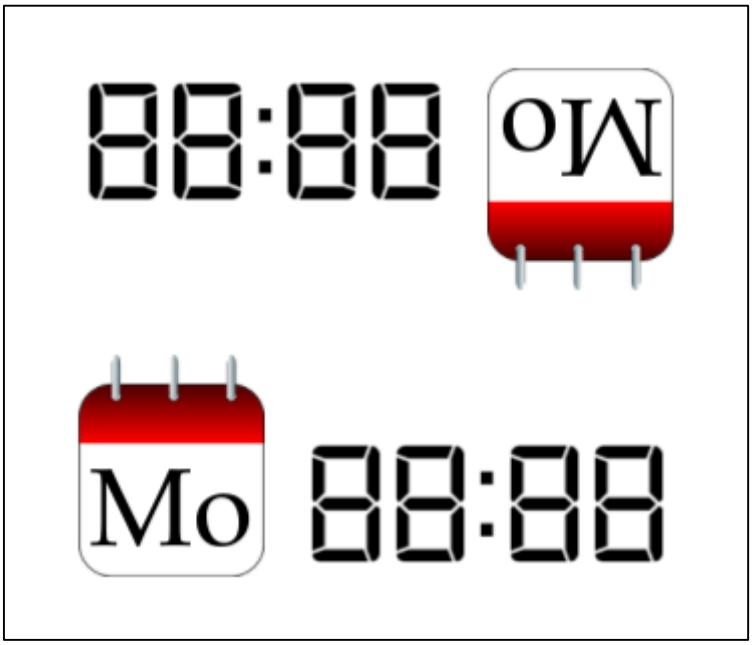
Prefab „EmployeePhotoSubgame“



Das für Ansicht von oben und von unten zweigeteilte Prefab „EmployeePhotoSubgame“ beinhält die grafische Darstellung des gleichnamigen Teilspiels, sowie als Komponente das Script mit gleichem Klassennamen.
Es zeigt zur Laufzeit immer eine Kombination aus einem Foto eines Hochschulangestellten und einem Namen eines Hochschulangestellten, der nicht zwangsläufig zum Foto passen muss.

TaskViewBottom	RectTransform mit Ein-/Ausblendeanimation für die Ansicht in Bildschirmausrichtung
TaskViewBottom/Photo	Image, das das Foto eines Hochschulangestellten anzeigt
TaskViewBottom/Name	Text mit Namen eines Hochschulangestellten als Inhalt
TaskViewTop	RectTransform mit Ein-/Ausblendeanimation für die um 180° gedrehte Ansicht, enthält dieselben Kindobjekte wie TaskViewBottom

Prefab „FlandernstrasseOpenSubgame“



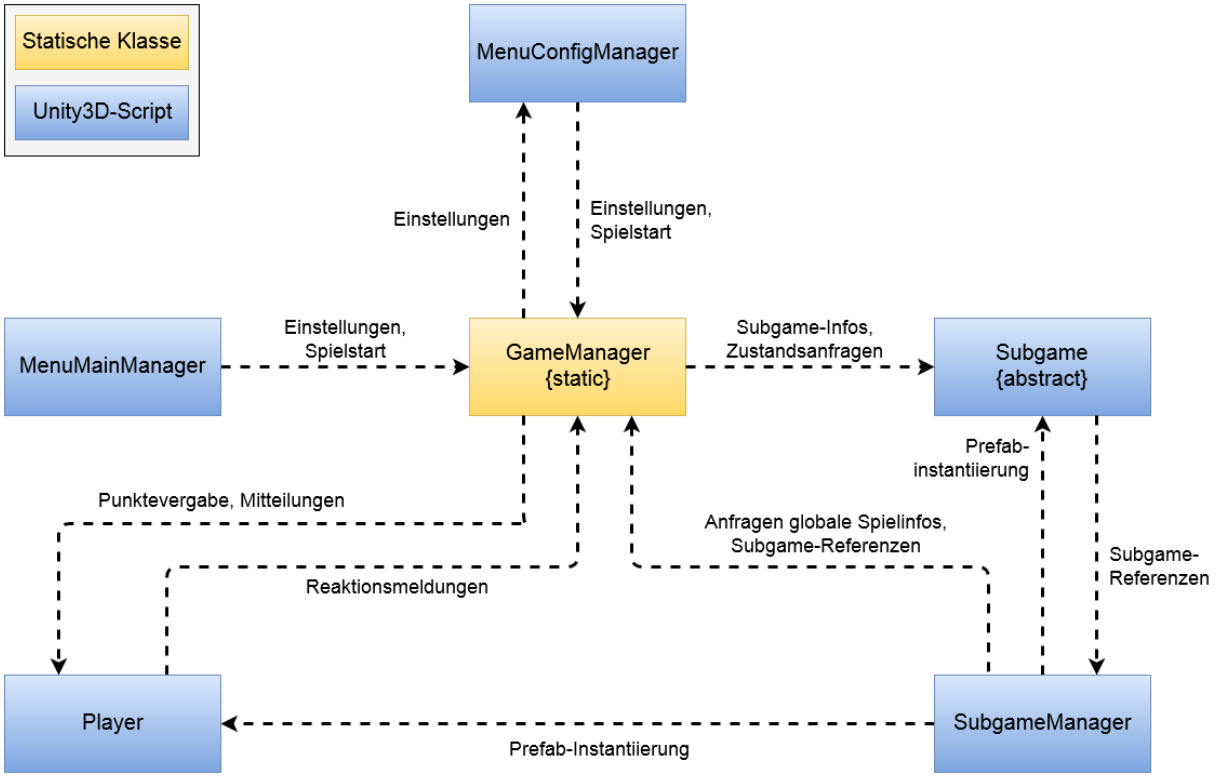
Das für Ansicht von oben und von unten zweigeteilte Prefab „FlandernstrasseOpenSubgame“ beinhält die grafische Darstellung des gleichnamigen Teilspiels, sowie als Komponente das Script mit gleichem Klassennamen.

Es zeigt immer einen zur Laufzeit veränderlichen Wochentag auf der linken Seite der Teilansicht, sowie eine sich kontinuierlich verändernde Uhrzeit auf der rechten Seite.

TaskViewBottom	RectTransform für die Ansicht in Bildschirmausrichtung
TaskViewBottom/Clock	Text, der die laufende Uhrzeit darstellt
TaskViewBottom/Calendar	RectTransform des Kalenderblatts mit veränderlicher Aufschrift
TaskViewBottom/Calendar/Background	Image des Kalenderblatts
TaskViewBottom/Calendar/Text	Text, der den laufenden Wochentag angibt
TaskViewTop	RectTransform für die um 180° gedrehte Ansicht, enthält dieselben Kindobjekte wie TaskViewBottom

Klassen

Architektur



Klasse „EmployeePhotoSubgame“

Basisklasse: TerminableTaskSubgame

Klasse, die als Script das Verhalten einer Instanz des EmployeePhotoSubgame-Prefabs implementiert.

`private enum ParsingState`
Zustände des Zustandsautomaten in der Methode LoadXML(), der die verschiedenen Elemente aus einer XML-Datei liest.

`private readonly string[] shownNamePaths`
Pfade der Text-Objekte der angezeigten Namen innerhalb des GameObjects dem das Skript als Komponente zugewiesen wurde.

`private readonly string[] shownPhotoPaths`
Pfade der Image-Objekte der angezeigten Fotos innerhalb des GameObjects dem das Skript als Komponente zugewiesen wurde.

`private readonly string[] blendAnimStateNames`
Namen der AnimationStates die beim Ein- und Ausblenden der aktuellen Aufgabe abgespielt werden.

`private Text[] shownTexts`
Text-Objekte, die den Namen eines Mitarbeiters anzeigen.

`private Image[] shownPhotos`
Image-Objekte, die das Foto des Mitarbeiters anzeigen.

`private List<string> loadedEmployeeNames`
Liste mit Namen aller mitarbeiter.

`private List<string> loadedPhotoPaths`
Pfade der Fotos aller Mitarbeiter innerhalb des ResourcesFolder des Subgames.

`private Employee trueEmployee`
Tatsächlich existierender Mitarbeiter (Name/Foto-Paar), der die Quelle des angezeigten Fotos darstellt.

`private void LoadXML()`
Lädt die loadedEmployeeNames und loadedPhotoNames aus der XML-Datei vom ResourcesFilePath des Subgames.

`private Employee LoadRandomEmployee()`
Lädt einen zufälliges Employee-Objekt aus den loadedEmployeeNames und loadedPhotoNames, das eine korrekte Name/Foto-Kombination enthält.

`private string LoadRandomName()`
Lädt einen zufälligen Namen aus den loadedEmployeeNames.

Klasse „FlandernstrasseOpenSubgame“

Basisklasse: Subgame

Klasse, die als Script das Verhalten einer Instanz des FlandernstrasseOpenSubgame-Prefabs implementiert.

`private enum ParsingState`
Zustände des Zustandsautomaten in der Methode LoadXML(), der die verschiedenen Elemente aus einer XML-Datei liest.

`private readonly string[] shownTimeViewPaths`
Pfade der Text-Objekte der angezeigten Uhrzeit innerhalb des GameObjects dem das Skript als Komponente zugewiesen wurde.

`private readonly string[] shownWeekdayViewPaths`
Pfade der Text-Objekte des angezeigten Wochentags innerhalb des GameObjects dem das Skript als Komponente zugewiesen wurde.

`private Text[] shownTimeViews`
Text-Objekte der angezeigten Uhrzeit.

`private Text[] shownWeekdayViews`
Text-Objekte des angezeigten Wochentags.

`private int determinedFontSize`
Erste automatisch eingestellte Schriftgröße der Uhrzeit der shownTimeViews.

`private DateTime currentTime`
NET DateTime, die die momentan anzuzeigende Uhrzeit enthält.

`private LinkedList<WeekbasedDateTimeInterval> openingHours;`
Liste der Zeitintervalle, in denen die Hochschule geöffnet ist.

`private void AdvanceTimeMins()`
Lasse die currentTime um eine Minute voranschreiten.

`private void SetRandomDateTime()`
Setze die currentTime auf eine zufällige Uhrzeit und Datum.

`private void UpdateTimeView()`
Aktualisiere die angezeigte Uhrzeit der shownTimeViews, sodass sie die currentTime widerspiegeln.

`private void UpdateWeekdayView()`
Aktualisiere den angezeigten Wochentag der shownWeekdayViews, sodass sie die currentTime widerspiegeln.

`private void LoadXML()`
Lädt die openingHours aus der XML-Datei vom ResourcesFilePath des Subgames.

Klasse „GameCompletionManager“

Klasse, die als Script auf Benutzereingaben in der „GameCompletion“-Szene reagiert und die Ausgabe der bereits berechneten Spielerstatistiken steuert.

`public const string sceneName`
Name der GameCompletion-Szene in Unity3D.

`public Text congratulationText`
Angezeigter Glückwunschs an den Gewinner des Spiels.

`public Text rankingText`
Angezeigte Rangliste der teilgenommenen Spieler.

`private readonly string[] congratulationPhraseSegments`
Abschnitte des Textes, die gemeinsam mit dem Spielernamen in congratulationText ausgegeben werden sollen.

`public void HandleMenuMainClick()`
Verarbeitet Klicken/Touch des Buttons zur Rückkehr ins Hauptmenü.

`public void HandleRestartGameClick()`
Verarbeitet Klicken/Touch des Buttons zum Neustart des Spiels.

`private void SetCongratulationText(LinkedList<PlayerStats> sourceRanking)`
Stellt den in congratulationText angezeigten Text mithilfe einer Liste von PlayerStats ein.

sourceRanking – Nach Punktanzahl von hoch nach niedrig sortierte LinkedList der PlayerStats der Teilnehmer.

`private void SetRankingText(LinkedList<PlayerStats> sourceRanking)`
Stellt den in rankingText angezeigten Text mithilfe einer Liste von PlayerStats ein.

sourceRanking – Nach Punktanzahl von hoch nach niedrig sortierte LinkedList der PlayerStats der Teilnehmer.

Klasse „GameManager“

Typ: statische Klasse

Global erreichbare Klasse, die Grundinformationen über die Einstellungen und den Aufbau des Spiels enthält. Dient als zentrales Vermittlungselement zwischen Spielteilnehmern und laufenden Teilspielen.

`public static List<string> subgames`
Liste der Namen der Prefabs der zu spielenden Teilspiele (Subgames).

`private const int maxPlayersCount`
Maximal einstellbare Spielerzahl (im aktuellen Layout 4).

`private static int playersCount`
Aktuell eingestellte Spieleranzahl.

`private static Subgame runningSubgame`
Referenz auf das Subgame-Script des aktuell laufenden Teilspiels.

`private static LinkedList<Player> participants`
Liste der Teilnehmer des Spiels. Mit jeweils einer Instanz des PlayerControls-Prefab verknüpfte Player-Scripts.

`private static LinkedList<PlayerStats> ranking`
Zuletzt berechnete, absteigend nach Punktzahl sortierte Rangliste der teilgenommenen Spieler.

`private static int subgameIndex`
Index des aktuell laufenden Teilspiels im Member subgames.

`public enum ResetOption`
Flags im als Bitset aufgebauten Parameter, der die durch die Methode Reset() zurückzusetzenden Parameter bestimmt.

`public static LinkedList<PlayerStats> Ranking{get}`
Zuletzt berechnete, absteigend nach Punktzahl sortierte Rangliste der teilgenommenen Spieler.

`public static Subgame RunningSubgame{set}`
Referenz auf Subgame-Script des aktuell laufenden Teilspiels.

`public static string NextSubgameName{get}`
Name des nächsten Subgames im Member subgames. Inkrementiert den subgameIndex.

`public static int PlayersCount{set; get}`
Aktuell eingestellte Spieleranzahl.

`public static void StartGame()`
Startet ein neues Spiel, falls der Member subgames Teilspiele enthält.

`public static void RegisterPlayer(Player registeringPlayer)`

Registriert einen Mitspieler, sodass dieser bei alle Spieler betreffenden Vorgängen berücksichtigt werden kann.

registeringPlayer – Script der Instanz des „PlayerControls“-Prefabs, das dem zu registrierenden Spieler gehört.

```
public static void UnregisterPlayer(Player unregisteringPlayer)
```

Meldet einen Mitspieler ab, sodass dieser fortan nicht mehr bei alle Spieler betreffenden Vorgängen berücksichtigt wird.

registeringPlayer – Script der Instanz des „PlayerControls“-Prefabs, das dem zu registrierenden Spieler gehört.

```
public static void OnPlayerReaction(Player reactor)
```

Verarbeitet die Reaktion eines Mitspielers auf die aktuell vom Teilspiel gestellte Aufgabe.

reactor - Script der Instanz des „PlayerControls“-Prefabs, das eine Reaktion erfasst hat.

```
public static void Reset(ResetOption resetSelection)
```

Setzt verschiedene Eigenschaften (nicht „Accessoren“) des Spiels auf die Ausgangswerte zurück.

resetSelection – Bitset, welches die in ResetOption beschriebenen Flags der zurückzusetzenden Eigenschaften enthält.

```
public static void ComputeRanking()
```

Berechnet die aktuelle, sortierte Rangliste nach absteigender Punktezahl und speichert diese im Member ranking.

Klasse „MenuConfigManager“

Script-Klasse für das MenuConfig-Menü. Verarbeitet Benutzereingaben über die dortigen Steuerelemente. Gibt die im GameManager gespeicherte Teilspielauswahl wieder und kann diesem Veränderungen übergeben.

`private enum ParsingState`
Zustände des Zustandsautomaten in der Methode BuildSubgameToggleList(), der die verschiedenen Elemente aus einer XML-Datei liest.

`public const string sceneName`
Name der MenuConfig-Szene in Unity3D.

`public RectTransform toggleGrid`
RectTransform-Komponente des GameObjects, in welchem die aus Checkbox und Teilspielbeschreibung bestehenden SubgameToggles platziert werden.

`public GameObject subgameTogglePrefab`
Prefab, auf dessen Basis die SubgameToggles instantiiert werden.

`private List<SubgameToggle> toggles`
Liste der instantiierten, und damit angezeigten, SubgameToggles.

`private const short visibleToggles`
Maximale Anzahl der SubgameToggles, sie ohne Scrollen sichtbar sein sollen. Resultiert in einer Größenänderung der einzelnen Toggles.

`private float toggleGridRectMinY`
Y-Wert des Minimum-Ankers des toggleGrids. Dieser verschiebt sich, wenn nicht alle SubgameToggles angezeigt werden können, damit mehr Raum zum Platzieren neuer Toggles frei wird, in welchen dann hineingescrollt werden kann.

`public void HandleGameStartClick()`
Verarbeitet Klicken/Touch des Buttons zum Starten des Spiels.

`public void HandleMenuMainClick()`
Verarbeitet Klicken/Touch des Buttons zur Rückkehr ins Hauptmenü.

`private void BuildSubgameToggleList()`
Belädt die Liste des Members toggles mit den im Dokument unter „StreamingAssets/subgame_list.xml“ angegebenen Teilspielen.

`private void PlaceSubgameToggles()`
Platziert die im Member toggles enthaltenen SubgameToggles im toggleGrid.

`private void SetSelectedToggles()`
Markiert die im GameManager gespeicherten Teilspiele in der angezeigten Liste als ausgewählt.

`private void SaveSelectedToggles()`
Speichert die in der angezeigten Liste ausgewählten Teilspiele im GameManager.

Klasse „MenuMainManager“

Script-Klasse für das MenuMain-Menü. Verarbeitet dort die Benutzereingaben für Spieleinstellung und -start über Steuerelemente und kommuniziert diese mit dem GameManager.

```
public const string sceneName
```
Name der MenuMain-Szene in Unity3D.

```
public Button ButtonPlayerCnt1
public Button ButtonPlayerCnt2
public Button ButtonPlayerCnt3
public Button ButtonPlayerCnt4
```
In der Szene enthaltene Buttons zur Auswahl der Spieleranzahl.

```
public void HandlePlayersCountClick(Button buttonClicked)
```
Verarbeitet Klicken/Touch einer der Buttons zur Auswahl der Spieleranzahl.

buttonClicked – Button, der betätigt wurde.

```
public void HandleStartGameClick()
```
Verarbeitet Klicken/Touch des Buttons zum Starten des Spiels.

```
public void HandleConfigGameClick()
```
Verarbeitet Klicken/Touch des Buttons zum Aufrufen des Einstellungsmenüs.

Klasse „Player“

Script-Klasse, die die Ein- und Ausgaben zu und vom „PlayerControls“-Prefab verarbeitet. Zeigt Meldungen für Mitteilungen durch den GameManager an und teilt diesem Eingaben vom Benutzer während des Spielgeschehens mit. Enthält statische (wie Name) und dynamische Informationen (Punktestand) über den einzelnen teilnehmenden Spieler.

`public Button buzzerButton`
Button, über den der Spieler reagieren kann, der im GameObject enthalten ist, auf das das Script angewendet wurde.

`public Text playerNameText`
Text-Objekt zur Anzeige des Spielernamens, das im GameObject enthalten ist, auf das das Script angewendet wurde.

`public Text notificationText`
Text-Objekt zur Anzeige von Benachrichtigungen an den Spieler, das im GameObject enthalten ist, auf das das Script angewendet wurde.

`public Text scoreText`
Text-Objekt zur Anzeige des aktuellen Punktestands, das im GameObject enthalten ist, auf das das Script angewendet wurde.

`public const int maxNameLength`
Maximal in playerNameText anzuzeigende Anzahl an Buchstaben des Spielernamens.

`private const int standardPoints`
Standardpunktezahl, die der Spieler für eine richtige Reaktion erhält oder für eine falsche Reaktion abgezogen bekommt.

`private readonly string[] successNotifications`
Menge an Erfolgsmeldungen, aus der dem Spieler bei korrekter Reaktion eine Meldung angezeigt werden kann.

`private readonly string[] failureNotifications`
s. successNotifications, für falsche Reaktionen.

`private enum BuzzerState`
Zustände, die der Buzzer besitzen kann. Ändert sich beispielsweise nach dessen Betätigung.

`private int score`
Aktuelle Punkteanzahl des Spielers.

`public string Name{get}`
Name des Spielers.

`public int Score{get}`
Aktuelle Punkteanzahl.

`public void HandleBuzzerButtonClick()`
Verarbeitet Klicken/Touch von buzzerButton.

`public void TakeReactionResult(bool wasCorrect)`
Nimmt das Ergebnis (Antwort des GameManagers) einer Reaktion auf die aktuelle Aufgabe des Spiels entgegen und veranlasst entsprechende Prozesse.

wasCorrect – War es korrekt, zu reagieren?

`public void ShowNotification(string notificationText)`
Zeigt dem Spieler über notificationText dauerhaft eine Nachricht an.

notificationText – Anzuzeigende Nachricht.

`public void ResetScore()`
Setzt den Punktestand des Spielers auf 0 zurück.

`public void ResetUI()`
Setzt das Aussehen der Steuerelemente des Spielers auf den Standardzustand während des Spiels zurück. Punkte und Name bleiben erhalten.

`public void InitUI(string initPlayerName)`
Initialisiert die Steuerelemente des Spielers auf die Standardwerte, setzt dabei den Spielernamen in playerNameText.

`private void UpdateScoreView()`
Überträgt den aktuellen Wert des Members score zur Anzeige in scoreText.

`private void ChangeScore(int signedDelta)`
Verändert den aktuellen Punktestand in score um den angegebenen Wert.

signedDelta – Vorzeichenbehaftete Änderung des Punktestands.

`private void ChangeBuzzerAppearance(BuzzerState state)`
Ändert das Erscheinungsbild der Steuerelemente in der Prefab-Instanz dem angegebenen Zustand entsprechend.

state – Neuer Zustand, auf den das Aussehen angepasst werden soll.

Klasse „PlayerStats“

Klasse, die Wertepaare bestehend aus einem Spielernamen und seiner Punktezahl darstellt. Dient dem einfachen Aufbau der Rangliste.

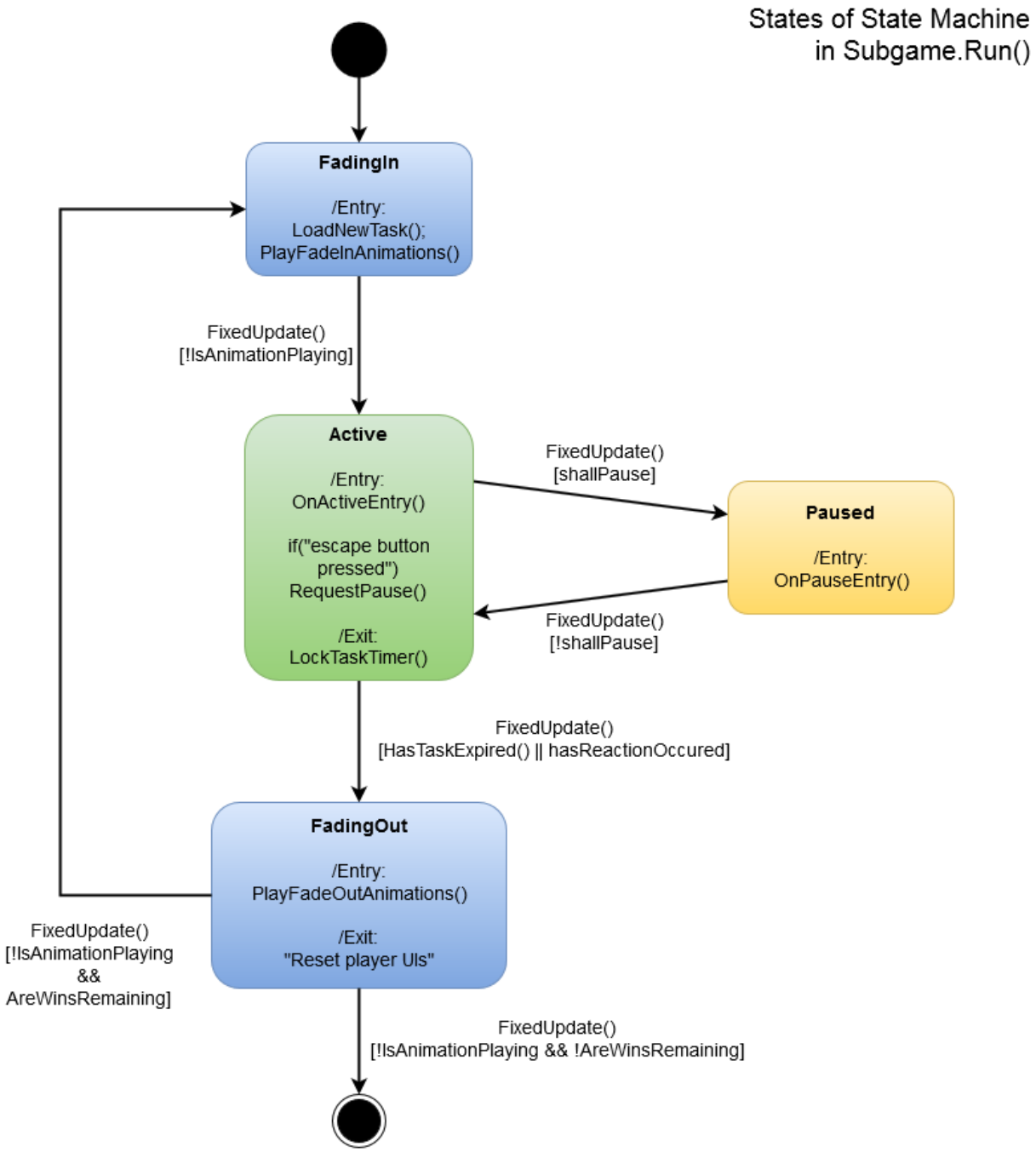
```
public readonly string name
```
Name des Spielers.

```
public readonly int score
```
Punktestand des Spielers.

```
public PlayerStats(string name, int score)
```
Erzeugt ein PlayerStats-Objekt mit dem angegebenen Namen und Punktestand.
Parameterbeschreibung s. Member-Attribute.

Klasse „Subgame“

Basisklasse aller Scripts, die das Verhalten der Prefabs der einzelnen Teilspiele auf abstrakter Ebene implementiert. Spielphasen innerhalb des Teilspiels finden sich in einem Zustandsautomaten wieder. Gibt den erbenden konkreten Teilspielen die Möglichkeit, ihr Verhalten für die verschiedenen Phasen zu definieren. Dient als Schnittstelle aller Teilspiele nach außen, z.B. zur Kommunikation mit dem GameManager.



`public enum SubgameState`
Mögliche Zustände des Zustandsautomaten in der Methode run().

`public enum InternalTrigger`
Mögliche Event-Trigger in der Methode run().

`private SubgameState currentSubgameState`
Aktueller Zustand des Zustandsautomaten in der Methode run().

`private SubgameState nextSubgameState`
Zustand, in dem sich der Zustandsautomat in der Methode run() im nächsten Durchlauf befinden wird.

`private InternalTrigger currentTrigger`
Aktueller Trigger in der Methode run().

`private Animation[] taskStartAnimations`
Animationen, die bei Start oder rückwärts bei Beendigung einer Aufgabe als Überblendung abgespielt werden sollen.

`protected static readonly string[] taskViewNames`
Namen der GameObjects der beiden Anzeigebereiche (oben und unten) für Aufgaben.

`protected const int taskViewsCount`
Anzahl der taskViews, im aktuellen Layout 2.

`protected bool  hasReactionOccured`
Gibt an, ob im aktuellen Schritt des Zustandsautomaten eine Spielerreaktion erfolgt ist.

`private const int requiredWins`
Anzahl der insgesamt von allen Spielern zusammen zu erreichende Anzahl korrekter Reaktionen (Nichtreaktionen ausgenommen) bis zum Fortschreiten zum nächsten Teilspiel.

`private int remainingWins`
Anzahl der verbleibenden korrekten Reaktionen aller Spieler zusammen bis zum Fortschreiten ins nächste Teilspiel (Nichtreaktionen ausgenommen).

`private bool shallPause`
Gibt an, ob sich der Zustandsautomat im Pausezustand befinden soll. Dient als Transition-Guard im Zustandsautomaten.

`public string ResourcesFilePath{get}`
Pfad zur Datei, die die zum Erstellen von Aufgaben nötigen Informationen, wie Zeichenketten, numerische Werte, Wahrheitswerte oder Pfade zu weiteren Ressourcen wie Fotos und Tondateien enthält.
Dieser lautet „StreamingAssets/[KonkreterKlassenname].xml“, wobei [KonkreterKlassenname] der Name des Scripts ist.

`public string ResourcesFolder{get}`
Verweist, die Datei unter dem ResourcesFilePath auf weitere Ressourcen, so sind diese im ResourcesFolder enthalten.
Dessen Pfad lautet ausgehend vom [Executable-Name]_Data-Ordner

„StreamingAssets/[KonkreterKlassenname]/“, wobei [KonkreterKlassenname] der Name des Scripts ist.

```
public bool ExpectsReaction{get}
```
Erwartet das Teilspiel aktuell die Reaktion eines Spielers auf die gestellte Aufgabe? (Schablonen“methode“)

```
public SubgameState State{get}
```
Aktueller Zustand des Zustandsautomaten, der den Verlauf des Teilspiels modelliert.

```
public bool ChangesState{get}
```
Gibt an, ob der Zustandsautomat des Subgames bei der nächsten Ausführung der Methode run() einen anderen Zustand als zuvor besitzen wird.

```
public bool HasEnded{get}
```
Gibt an, ob das Teilspiel vom Zustandsautomaten beendet wurde (schließt das Ende aller Ausblendeanimationen ein).

```
protected Animation[] TaskStartAnimations{get}
```
Animationen, die bei Start oder rückwärts bei Beendigung einer Aufgabe als Überblendung abgespielt werden sollen.

```
protected bool AreWinsRemaining{get}
```
Stehen weitere korrekte Spielerreaktionen (keine Nichtreaktionen) bis zur Beendigung des Teilspiels aus?

```
private bool IsAnimationPlaying{get}
```
Gibt an, ob im Moment eine Übergangsanimation des Teilspiels abgespielt wird.

```
public void Run()
```
Setzt den Zustandsautomaten um einen Schritt fort.

```
public void RequestPause()
```
Beantragt beim Zustandsautomaten das Pausieren des Spiels.

```
private void EntryActivity()
```
Bestimmt, welche Aktivitäten bei Eintritt in den aktuellen Zustand ausgeführt werden müssen und veranlasst diese.

```
private void DoActivity()
```
Bestimmt, welche Aktivitäten während des aktuellen Zustands ausgeführt werden müssen und veranlasst diese.

```
private void ExitActivity()
```
Bestimmt, welche Aktivitäten bei Austritt aus dem aktuellen Zustand ausgeführt werden müssen und veranlasst diese.

```
private bool TryTransition()
```
Überprüft ob unter den aktuellen Zuständen aller Transition-Guards ein Zustand erfolgt und setzt den nextSubgameState.

Rückgabewert – Hat der Übergang in einen neuen Zustand stattgefunden?

`protected abstract void PlayFadeInAnimations()`
Gibt den erbenden konkreten Subgames die Möglichkeit, Animationen, die sie dem Member taskStartAnimations zugewiesen haben, abzuspielen, um ihre aktuelle Aufgabe einzublenden.

`protected abstract void PlayFadeOutAnimations()`
Gibt den erbenden konkreten Subgames die Möglichkeit, Animationen, die sie dem Member taskStartAnimations zugewiesen haben, abzuspielen, um ihre aktuelle Aufgabe auszublenden.

`public void DestroyObject()`
Veranlasst die Zerstörung der Instanz des Prefabs, der das Script zugewiesen wurde.

`private void LoadNewTask()`
Lädt Informationen für eine neue Aufgabe in die grafischen Elemente von Unity3D. Ruft für weitere in der Unterklasse ausgeführten Schritte OnLoadNewTask() auf. (Schablonenmethode)

`protected abstract void OnLoadNewTask()`
Lädt Informationen für eine neue Aufgabe in die grafischen Elemente von Unity3D, die nur durch die Unterklasse bereitgestellt werden können. (LoadNewTask() als Schablonenmethode)

`protected void DecreaseRemainingWins()`
Dekrementiert die Anzahl der verbleibenden nötigen korrekten Reaktionen (nicht korrekte Nichtreaktionen) bis zum Ende des Teilspiels.

`protected abstract bool HasTaskExpired()`
Ist die aktuell angezeigte Aufgabe abgelaufen, weil kein Spieler reagiert hat?

Rückgabewert – Ist die Aufgabe abgelaufen?

`protected abstract bool OnExpectsReaction()`
Gibt durch Schritte in der Unterklasse Auskunft darüber, ob diese im Augenblick eine Reaktion vom Spieler erwartet. (ExpectsReaction-Eigenschaft als Schablonen“methode“)

Rückgabewert – Erwartet das Teilspiel eine Reaktion?

`protected abstract void OnActiveEntry()`
Schritte, die in der Unterklasse bei Eintritt in den Zustand “Active” ausgeführt werden sollen.

`private void PauseEntry()`
Schritte, die beim Eintritt in den Zustand „Paused“ ausgeführt werden sollen. (Schablonenmethode)

`protected abstract void OnPauseEntry()`
Schritte, die in der Unterklasse bei Eintritt in den Zustand “Paused” ausgeführt werden sollen. (PauseEntry() als Schablonenmethode)

`protected abstract void OnActiveExit()`
Schritte, die in der Unterklasse bei Austritt aus dem Zustand “Active” ausgeführt werden sollen.

`protected abstract void OnFadingInExit()`
Schritte, die in der Unterklasse bei Austritt aus dem Zustand “FadingIn” ausgeführt werden sollen.

Klasse „SubgameManager“

Beinhält als Script für die Subgame-Szene die Logik für Aufruf und Anzeige der im GameManager festgelegten Teilspiele, sowie die Erzeugung der Spieler und ihrer grafischen Steuerelemente. Verwirklicht dadurch und durch das zur Verfügung stellen einer Pausefunktion die oberste Ausführungsebene des Spielflusses.

`public const string sceneName`
Name der Subgame-Szene in Unity3D.

`public RectTransform pauseDialogueBottom`
RectTransform des Dialogfensters, das bei Pausierung in der unteren Bildschirmhälfte angezeigt werden soll.

`public RectTransform pauseDialogueTop`
RectTransform des Dialogfensters, das bei Pausierung in der oberen Bildschirmhälfte angezeigt werden soll.

`public RectTransform playerCtrlAreaBottom`
RectTransform, in dem die unteren Steuerelemente für Mitspieler eingeblendet werden.

`public RectTransform playerCtrlAreaTop`
RectTransform, in dem die oberen Steuerelemente für Mitspieler eingeblendet werden.

`public RectTransform taskZone`
Bereich, in dem Aufgaben erscheinen, d.h. in dem die Instanz des Subgame-Prefabs platziert wird.

`public GameObject playerControlsPrefab`
Prefab, das für die Steuerelemente der einzelnen Spieler benutzt wird.

`private bool wasValidSubgameInstantiated`
Gibt an, ob seit Laden der Szene bereits ein Subgame-Prefab instantiiert wurde.

`private Subgame runningSubgame`
Script der Instanz des aktuell platzierten Teilspiels.

`private const float ctrlsAnchorMaxY`
Y-Wert des Maximum-Ankers des RectTransforms der Instanzen des Prefabs für die Steuerelemente der Spieler innerhalb von playerCtrlAreaBottom und der um 180° gedrehten Instanzen in playerCtrlAreaTop.

`private const float pauseDialogueAlpha`
Alphakanal aller Farben der Pausedialogfenster, wenn dieses sichtbar sein soll.

`private const string subgamesResFolder`
Name des Ordners innerhalb von Resources, in dem sich alle Prefabs zur Instantiierung der Teilspiele befinden.

`private enum BuzzerPosition`
Mögliche diskrete Positionen der Steuerelemente für Spieler im aktuell festgelegten Layout.

```
public Subgame PlaceSubgamePrefab(string subgamePrefabName)
```

Platziere die Instanz des Prefabs mit dem angegebenen Namen in der taskZone.

subgamePrefabName –	Name des zu instantiierenden Prefabs
Rückgabewert –	Subgame-Script der platzierten Instanz, null wenn Instantiierung nicht erfolgen konnte.

```
private void PlacePlayerControls()
```

Platziere so viele Instanzen des Prefabs für Mitspieler-Steuerelemente in den dafür bestimmten Bereichen, wie Spieler teilnehmen sollen.

```
private void PlaceNextSubgamePrefab()
```

Platziere das nächste Prefab des Teilspiels in der taskZone, das sich in der getroffenen Auswahl an Teilspielen befindet. Kann das Prefab nicht instantiiert werden, wird solange versucht, das darauffolgende Teilspiel zu platzieren, wie weitere übrig sind.

```
private void ShowPauseDialogues()
```

Zeige die Pausedialogfenster an.

```
private void HidePauseDialogues()
```

Verstecke die Pausedialogfenster.

Klasse „TerminableTaskSubgame“

Basisklasse: Subgame

Basisklasse aller Scripts für Teilspiele, deren Aufgaben die für die Korrektheit einer Spielerreaktion entscheidende Eigenschaft über die gesamte Zeit beibehalten und nach einer gewissen Zeit selbständig durch eine neue Aufgabe getauscht werden. Stellt den erbenden Teilspiel-Scripts einen steuerbaren Timer mit festgelegter Laufzeit zur Verfügung.

`private const float taskDuration`
Dauer einer angezeigten Aufgabe im Zustand „Active“ in Sekunden.

`private float taskRuntime`
Gibt an, wie lange der TaskTimer schon läuft, d.h. wie lange sich das Subgame bereits im Zustand „Active“ befindet.

`private bool isTaskTimerRunning`
Gibt an, ob der TaskTimer im Moment läuft.

`private float TaskRuntime{get}`
Gibt an, wie lange der TaskTimer schon läuft, d.h. wie lange sich das Subgame bereits im Zustand „Active“ befindet.

`private void InitTaskTimers()`
Setze den TaskTimer zurück, um ihn auf ein erstes oder erneutes Starten vorzubereiten.

`private void ReleaseTaskTimer()`
Starte den TaskTimer oder setze ihn fort.

`private void LockTaskTimer()`
Halte den TaskTimer an.

`private void TriggerTaskTimer()`
Teile dem TaskTimer mit, dass die Zeit Time.fixedDeltaTime vergangen ist. Ohne Aufruf ist der TaskTimer ohne Funktion.

Klasse „WeekbasedDateTimeInterval“

Stellt mithilfe der DateTime-Struktur des NET-Frameworks ein Zeitintervall zwischen zwei Uhrzeiten dar, der sich über einen Zeitraum innerhalb der über die Wochentage definierten Woche erstreckt.

`public const int minutesInWeek`
Minuten, die in einer Woche vergehen.

`private const Int32 baseYear`
Jahr, das als Basis für die wochenbasierten Tage verwendet wird.

`private const Int32 baseMonth`
Monat, der als Basis für die wochenbasierten Tage verwendet wird.

`private const Int32 baseSeconds`
Sekundenwert, der jede Uhrzeit erhält.

`private DateTime start`
Zeitpunkt, der den Beginn des Intervalls angibt.

`private DateTime end`
Zeitpunkt, der das Ende des Intervalls angibt.

`public WeekbasedDateTimeInterval(DayOfWeek startDay, Int32 startHours,`
`Int32 startMinutes, DayOfWeek endDay, Int32 endHours, Int32 endMinutes)`
Erzeugt ein neues Zeitintervall innerhalb einer Woche.

startDay – Wochentag des Startzeitpunkts.
startHours – Stundenzahl der Uhrzeit des Startzeitpunkts.
startMinutes – Minutenzahl der Uhrzeit des Startzeitpunks.
endDay – Wochentag des Endzeitpunkts.
endHours – Stundenzahl der Uhrzeit des Endzeitpunkts.
endMinutes – Minutenzahl der Uhrzeit des Endzeitpunkts.

`public bool Contains(DateTime checkingDateTime)`
Berechnet, ob der angegebene Zeitpunkt sich auf Wochentag und Uhrzeit bezogen im Intervall befindet.

checkingDateTime – Zu überprüfender Zeitpunkt.
Rückgabewert – Befindet sich der Zeitpunkt im Intervall?

`private static Int32 ToIntBaseMonday(DayOfWeek convertingDay)`
Gibt den Index des angegebenen Wochentags in einer mit Montag startenden Woche zurück.

convertingDay – Wochentag, dessen Index gesucht ist.

Einbinden neuer bzw. Änderung vorhandener Teilspiele

Benennung

Der Name des Teilspiels sollte als einziges Wort in Camel-Case-Schreibweise vorliegen. Daraus folgen die weiteren Namen für:

Prefab	[Teilspielname] im Dateisystem [Teilspielname].prefab
Script	[Teilspielname].cs mit Klasse [Teilspielname]
XML-Datei für Aufgabeninhalte	[Teilspielname].xml
Ordner für Assets der Aufgabeninhalte	[Teilspielname]

Prefab als Basis

Teilspiele werden als Prefab implementiert, das als Komponente des RectTransforms, das eine Aufgabe enthält, eine Animation zum Überblenden zwischen verschiedenen Aufgaben bereitstellen kann. Die Logik des Teilspiels wird als Script-Komponente des Prefabs realisiert. Die Klasse dieses Scripts muss von Subgame, oder falls jede Aufgabe nach einer gewissen Zeit ohne Spielerreaktion enden soll, von TerminableTaskSubgame erben und den
Dieses Prefab muss in folgenden Ordner gelegt werden:

[Unity3D-Projekt]/Assets/Resources/SubgamePrefabs

Das Script selbst findet seinen Platz im Projektordner unter

[Unity3D-Projekt]/Assets/Scripts

Informationen zur Erstellung von Aufgaben

Soll das Spiel flexibel erweiter- oder veränderbar sein, so kann dies im Script durch Laden der zur Erstellung der verschiedenen Aufgaben benötigten Informationen aus einer XML-Datei erfolgen. Diese kann direkt zu ladende Werte wie Strings, numerische oder Wahrheitswerte enthalten oder Pfade weiterer Assets wie Bilder oder Tondateien. Der Pfad dieser Datei muss lauten:

[Unity3D-Projekt]/Assets/StreamingAssets/[Teilspielname].xml

Nach dem Build des Projekts lautet der Pfad automatisch:

[Build-Pfad]/HSES_Reaktor_Data/StreamingAssets/[Teilspielname].xml

Erweiterbarer Assets-Pool

Enthält die Datei Pfade weiterer Assets, so sollten diese in folgendem neu zu erstellenden Ordner platziert werden:

[Unity3D-Projekt]/Assets/StreamingAssets/[Teilspielname]

Nach dem Build des Projekts lautet der Pfad automatisch:

[Build-Pfad]/HSES_Reaktor_Data/StreamingAssets/[Teilspielname]

Erweiterung der Liste verfügbarer Teilspiele

Damit der HS-ES-Reaktor das neue Teilspiel erkennen kann, muss es in der Datei unter

[Unity3D-Projekt]/Assets/StreamingAssets/SubgameList.xml

vermerkt werden.
Dies kann auch nach dem Build des Projekts geschehen, der Pfad lautet dann automatisch:

[Build-Pfad]/HSES_Reaktor_Data/StreamingAssets/SubgameList.xml

Der Eintrag hat nach folgendem Schema zu geschehen:

```
<subgame>
    <prefab>[Teilspielname]</prefab>
    <menuName>[Kurzbeschreibung]</menuName>
</subgame>
```

Die Kurzbeschreibung wird im Menü der auswählbaren Spiele angezeigt werden (siehe auch Kapitel Prefab „SubgameToggle“ bzw. Szene „MenuConfig“).
Dies sollte bei der Wahl der Länge der Beschreibung bedacht werden.