

Grau en Enginyeria Informàtica de Gestió i Sistemes d’Informació

Desenvolupament d’un Framework per programar Utility-Based AI

Annexos

Jordi Romagosa i Mellado

Tutor: Enric Sesa i Nogueras

Curs 2017- 2018



**Introducció**

L’objectiu d’aquests annexes és proporcionar la documentació necessària per facilitar tant la lectura de la memòria com la valoració de la codificació. S’han agrupat les diverses taules i figures de la memòria per facilitar-ne la consulta i per accedir amb facilitat a la informació en els diversos apartats de la memòria, en cas que el lector ho necessiti. Només s’han incorporat les figures i taules rellevants pel disseny i programació tant del framework com de la simulació. Pel mateix motiu, també s’ha fet el recull del codi de programació tant del framework com dels scripts i classes de la simulació.

Índex

[1 Annex I - Recull de figures i taules 3](#_Toc516241092)

[1.1 Recull de figures i taules del framework 3](#_Toc516241093)

[1.1.1 Figura 7.1 Diagrama de classes del framework 3](#_Toc516241094)

[1.1.2 Taula 7.1 Classes que hereten de UtilityFuncion 4](#_Toc516241095)

[1.1.3 Figura 7.2 Diagrama de seqüència del framework 5](#_Toc516241096)

[1.2 Recull de figures i taules de la simulació 5](#_Toc516241097)

[1.2.1 Taula 9.1 Atributs del personatge 5](#_Toc516241098)

[1.2.2 Taula 9.2 Accions del personatge 6](#_Toc516241099)

[1.2.3 Figura 9.5 Diagrama de classes de la simulació 7](#_Toc516241100)

[2 Annex II - Codificació 9](#_Toc516241101)

[2.1 Codificació del framework 9](#_Toc516241102)

[2.1.1 Action 9](#_Toc516241103)

[2.1.2 Context 9](#_Toc516241104)

[2.1.3 BestActionCalculation 10](#_Toc516241105)

[2.1.4 UtilityFunction 12](#_Toc516241106)

[2.1.5 ConstantUtilityFunction 12](#_Toc516241107)

[2.1.6 ComposedUtilityFunction 13](#_Toc516241108)

[2.1.7 LinearUtilityFunction 13](#_Toc516241109)

[2.1.8 LogisticUtilityFunction 14](#_Toc516241110)

[2.1.9 LogitUtilityFunction 15](#_Toc516241111)

[2.1.10 QuadraticUtilityFunction 16](#_Toc516241112)

[2.2 Codificació de la simulació 17](#_Toc516241113)

[2.2.1 ActionInformation 17](#_Toc516241114)

[2.2.2 Atribute 17](#_Toc516241115)

[2.2.3 GameController 19](#_Toc516241116)

[2.2.4 GameState 23](#_Toc516241117)

[2.2.5 UIAtribute 24](#_Toc516241118)

[2.2.6 UIController 24](#_Toc516241119)

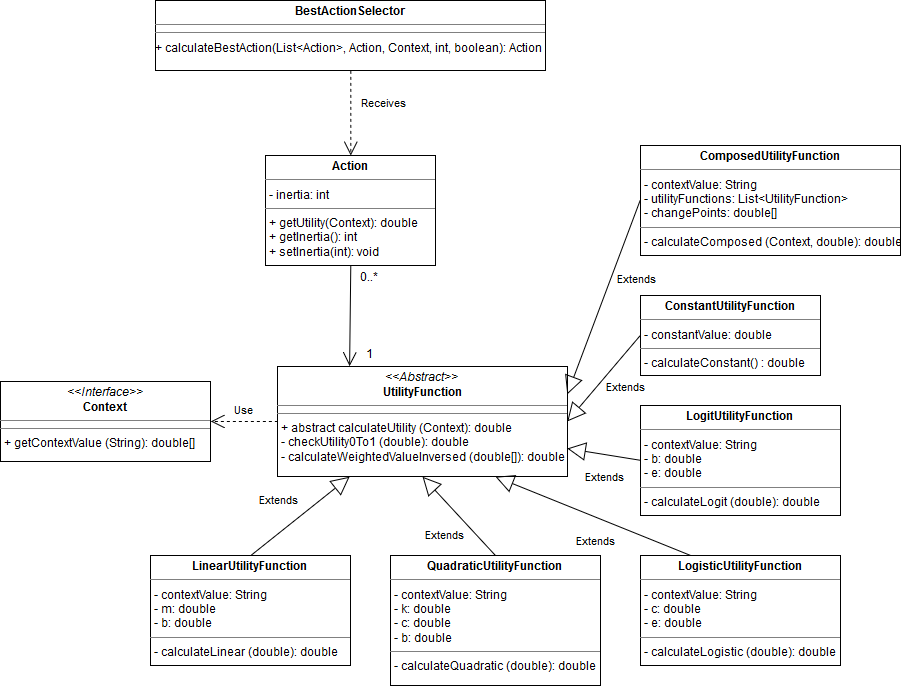
[3 Annex III - Contingut del CD 27](#_Toc516241120)

# Annex I - Recull de figures i taules

En aquesta secció es mostra el recull de figures i taules de la memòria que s’ha considerat que són prou rellevants com perquè el lector hi hagi de poder accedir amb facilitat. Les figures i taules apareixen classificades i en el mateix ordre que apareixen en la memòria. S’han numerat seguint la numeració de la memòria per mantenir la relació.

## Recull de figures i taules del framework

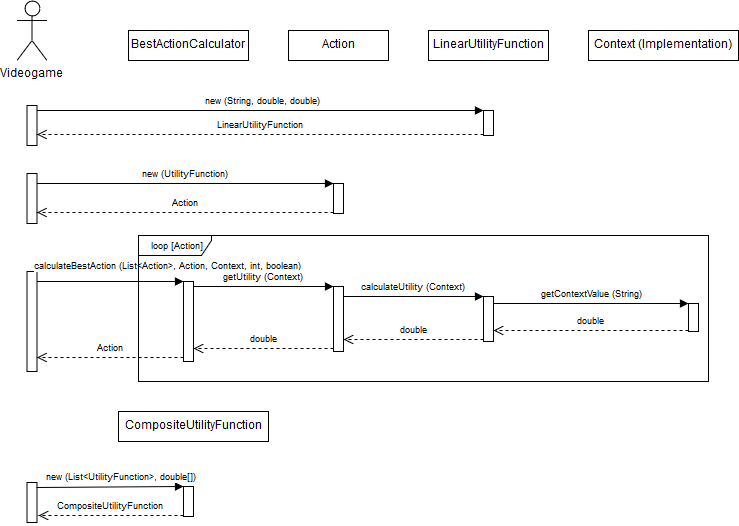
### Figura 7.1 Diagrama de classes del framework



### Taula 7.1 Classes que hereten de UtilityFuncion

|  |  |  |
| --- | --- | --- |
| **Nom** | **Funció de càlcul** | **Representació gràfica** |
| Linear |  |  |
| Quadratic |  |  |
| Constant |  |  |
| Logistic |  |  |
| Logit |  |  |
| Composed |  |  |

### Figura 7.2 Diagrama de seqüència del framework



## Recull de figures i taules de la simulació

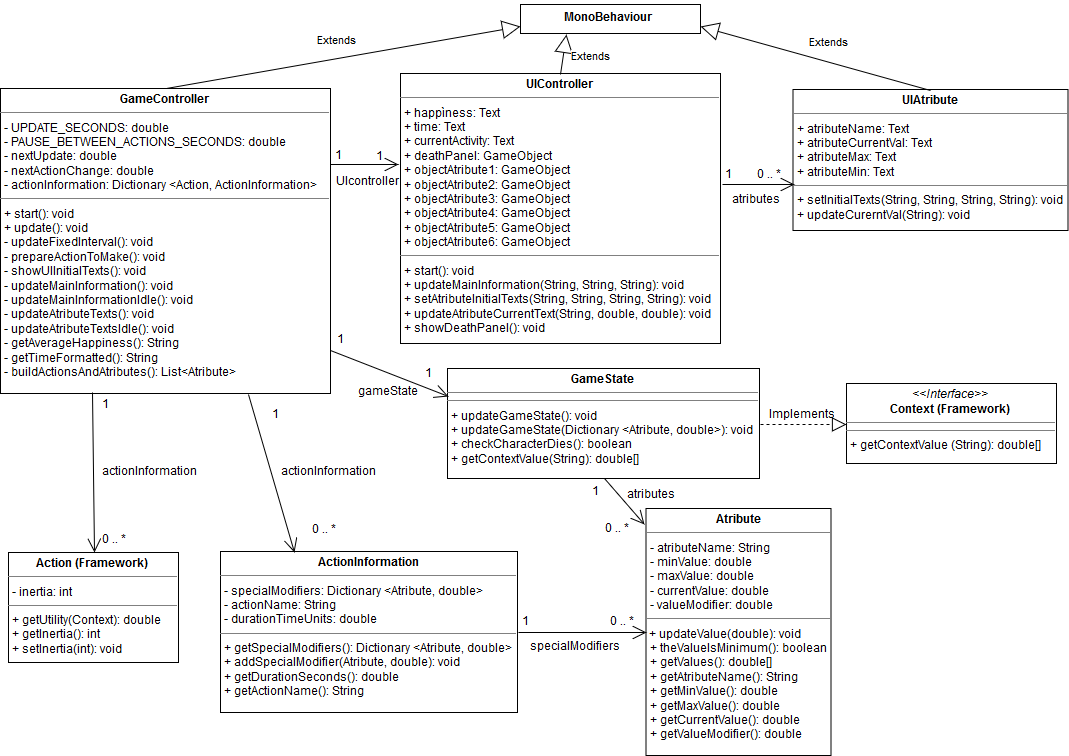
### Taula 9.1 Atributs del personatge

|  |  |  |  |
| --- | --- | --- | --- |
| **Atribut** | **Màxim** | **Mínim** | **Modificador** |
| Fullness | 100 | 0 | -3 |
| Awake | 100 | 0 | -1 |
| Relax | 100 | 0 | -1 |
| Clean | 100 | 0 | -1 |
| Money | 100 | 0 | -2 |
| Energy | 100 | 0 | +1 |

### Taula 9.2 Accions del personatge

|  |  |  |  |
| --- | --- | --- | --- |
| **Acció** | **Unitats de Temps** | **Atribut** | **Modificador** |
| Eat | 4 | Fullness | +10 |
| Sleep | 8 | Awake | +5 |
| Relax | +1 |
| Work | 6 | Money | +8 |
| Relax | -4 |
| Exercise | 4 | Relax | +15 |
| Clean | -5 |
| Energy | -10 |
| Watch TV | 2 | Relax | +10 |
| Shower | 3 | Clean | +40 |

### Figura 9.5 Diagrama de classes de la simulació



# Annex II - Codificació

En aquesta secció s’ha escrit la codificació del framework i de la simulació, per poder consultar en cas de necessitat durant la lectura de la memòria. S’ha decidit escriure la codificació sencera degut a que aquesta no és massa extensa.

## Codificació del framework

### Action

using System;

namespace UtilityAI

{

public class Action

{

private double inertia;

private UtilityFunction utilityFunction;

public Action(double inertia, UtilityFunction utlityFunction)

{

this.inertia = inertia;

this.utilityFunction = utlityFunction;

}

public Action(UtilityFunction utilityFunction) : this(0, utilityFunction) { }

public double Inertia { get => inertia; set => inertia = value; }

public double GetUtility(Context context)

{

return utilityFunction.CalculateUtility(context);

}

}

}

### Context

using System;

namespace UtilityAI

{

public interface Context

{

double[] GetContextValue(String valueName);

}

}

### BestActionCalculation

using System;

using System.Collections.Generic;

using System.Linq;

namespace UtilityAI

{

public class BestActionCalculator

{

public const int BEST\_ACTION = 0;

public const int WEIGHTED\_RANDOM = 1;

public const int RANDOM\_BEST\_3 = 2;

public static Action CalculateBestAction(List<Action> actions, Action currentAction, Context context, int chooseMethod, Boolean useInertia)

{

switch (chooseMethod)

{

case BEST\_ACTION:

return BestActionMethod(actions, currentAction, context, useInertia);

case WEIGHTED\_RANDOM:

return WeightedRandomMethod(actions, currentAction, context, useInertia);

case RANDOM\_BEST\_3:

return RandomBest3Method(actions, currentAction, context, useInertia);

default:

return DefaultActionChoose(actions, currentAction);

}

}

private static Action DefaultActionChoose(List<Action> actions, Action currentAction)

{

if (currentAction != null) return currentAction;

int randomIndex = new Random().Next(actions.Count);

return actions[randomIndex];

}

private static double ApplyInertiaToUtility(double baseUtility, double inertiaValue)

{

if (inertiaValue > 1) inertiaValue = 1;

else if (inertiaValue < 0) inertiaValue = 0;

return baseUtility + baseUtility \* inertiaValue;

}

private static Action BestActionMethod(List<Action> actions, Action currentAction, Context context, Boolean useInertia)

{

Action bestAction = actions[0];

actions.RemoveAt(0);

double bestUtility = bestAction.GetUtility(context);

if (useInertia && bestAction.Equals(currentAction))

bestUtility = ApplyInertiaToUtility(bestUtility, bestAction.Inertia);

foreach (Action action in actions){

double currentActionUtility = action.GetUtility(context);

if (useInertia && action.Equals(currentAction))

currentActionUtility = ApplyInertiaToUtility(currentActionUtility, action.Inertia);

if (currentActionUtility > bestUtility)

{

bestAction = action;

bestUtility = currentActionUtility;

}

}

return bestAction;

}

private static Action WeightedRandomMethod(List<Action> actions, Action currentAction, Context context, Boolean useInertia)

{

List<KeyValuePair<Action, double>> actionUtilityPairs = new List<KeyValuePair<Action, double>>();

double accumulatedUtility = 0;

foreach (Action action in actions)

{

double currentActionUtility = action.GetUtility(context);

if (useInertia && action.Equals(currentAction))

currentActionUtility = ApplyInertiaToUtility(currentActionUtility, action.Inertia);

accumulatedUtility += currentActionUtility;

actionUtilityPairs.Add(new KeyValuePair<Action, double>(action, accumulatedUtility));

}

double random = new Random().NextDouble() \* accumulatedUtility;

foreach (KeyValuePair<Action, double> actionUtilityPair in actionUtilityPairs)

{

if (random < actionUtilityPair.Value)

return actionUtilityPair.Key;

}

return currentAction;

}

private static Action RandomBest3Method(List<Action> actions, Action currentAction, Context context, Boolean useInertia)

{

Dictionary<Action, double> actionUtilityPairs = new Dictionary<Action, double>();

foreach (Action action in actions) {

double currentActionUtility = action.GetUtility(context);

if (useInertia && action.Equals(currentAction))

currentActionUtility = ApplyInertiaToUtility(currentActionUtility, action.Inertia);

actionUtilityPairs.Add(action, currentActionUtility);

}

var items = from pair in actionUtilityPairs orderby pair.Value descending select pair;

Dictionary<Action, double> bestActionUtilityPairs = new Dictionary<Action, double>();

foreach (KeyValuePair<Action, double> pair in items)

{

if (bestActionUtilityPairs.Count > 3)

break;

bestActionUtilityPairs.Add(pair.Key, pair.Value);

}

return bestActionUtilityPairs.ElementAt(new Random().Next(0, bestActionUtilityPairs.Count)).Key;

}

}

}

### UtilityFunction

using System;

namespace UtilityAI

{

public abstract class UtilityFunction

{

public abstract double CalculateUtility(Context context);

protected double CalculateWeightedValueInversed(double[] values)

{

double maxValue = values[0];

double minValue = values[1];

double currentValue = values[2];

double weightedValue = (currentValue - minValue) / (maxValue - minValue);

return -weightedValue + 1;

}

protected double CheckUtility0To1(double utility)

{

if (utility > 1) return 1;

else if (utility < 0) return 0;

return utility;

}

}

}

### ConstantUtilityFunction

using System;

using System.Collections.Generic;

using System.Text;

namespace UtilityAI.UtilityFunctionImplementations

{

public class ConstantUtilityFunction : UtilityFunction

{

double constantValue;

public ConstantUtilityFunction(double constantValue)

{

this.constantValue = constantValue;

}

public override double CalculateUtility(Context context)

{

double utility = CalculateConstant();

return CheckUtility0To1(utility);

}

private double CalculateConstant()

{

return constantValue;

}

}

}

### ComposedUtilityFunction

using System;

using System.Collections.Generic;

using System.Linq;

namespace UtilityAI.UtilityFunctionImplementations

{

public class ComposedUtilityFunction : UtilityFunction

{

private String contextValue;

private List<UtilityFunction> utilityFunctions;

private double[] changePoints;

public ComposedUtilityFunction(string contextValue, List<UtilityFunction> utilityFunctions, double[] changePoints)

{

this.contextValue = contextValue;

this.utilityFunctions = utilityFunctions;

this.changePoints = changePoints;

}

public override double CalculateUtility(Context context)

{

double[] valueState = context.GetContextValue(contextValue);

double currentWeightedValue = CalculateWeightedValueInversed(valueState);

double utility = CalculateComposed(context, currentWeightedValue);

return CheckUtility0To1(utility);

}

private double CalculateComposed(Context context, double x)

{

int functionChosen = 0;

foreach (double n in changePoints)

{

if (x < n) break;

else functionChosen++;

}

return utilityFunctions.ElementAt(functionChosen).CalculateUtility(context);

}

}

}

### LinearUtilityFunction

using System;

namespace UtilityAI.UtilityFunctionImplementations

{

public class LinearUtilityFunction : UtilityFunction

{

private String contextValue;

private double m;

private double b;

public LinearUtilityFunction(string contextValue, double m, double b)

{

this.contextValue = contextValue;

this.m = m;

this.b = b;

}

public override double CalculateUtility(Context context)

{

double[] valueState = context.GetContextValue(contextValue);

double currentWeightedValue = CalculateWeightedValueInversed(valueState);

double utility = CalculateLinear(currentWeightedValue);

return CheckUtility0To1(utility);

}

private double CalculateLinear(double x)

{

return m \* x + b;

}

}

}

### LogisticUtilityFunction

using System;

namespace UtilityAI.UtilityFunctionImplementations

{

public class LogisticUtilityFunction : UtilityFunction

{

private String contextValue;

private double c;

private double e;

public LogisticUtilityFunction(string contextValue, double c, double e)

{

this.contextValue = contextValue;

this.c = c;

this.e = e;

}

public override double CalculateUtility(Context context)

{

double[] valueState = context.GetContextValue(contextValue);

double currentWeightedValue = CalculateWeightedValueInversed(valueState);

double utility = CalculateLogistic(currentWeightedValue);

return CheckUtility0To1(utility);

}

private double CalculateLogistic(double x)

{

double power = -x + c;

return 1 / (1 + Math.Pow(e, power));

}

}

}

### LogitUtilityFunction

using System;

using System.Collections.Generic;

using System.Text;

namespace UtilityAI.UtilityFunctionImplementations

{

public class LogitUtilityFunction : UtilityFunction

{

private String contextValue;

private double b;

private double e;

public LogitUtilityFunction(string contextValue, double b, double e)

{

this.contextValue = contextValue;

this.b = b;

this.e = e;

}

public override double CalculateUtility(Context context)

{

double[] valueState = context.GetContextValue(contextValue);

double currentWeightedValue = CalculateWeightedValueInversed(valueState);

double utility = CalculateLogit(currentWeightedValue);

return CheckUtility0To1(utility);

}

private double CalculateLogit(double x)

{

double number = x / (1 - x);

return Math.Log(number, e) + b;

}

}

}

### QuadraticUtilityFunction

using System;

namespace UtilityAI.UtilityFunctionImplementations

{

public class QuadraticUtilityFunction : UtilityFunction

{

private String contextValue;

private double k;

private double c;

private double b;

public QuadraticUtilityFunction(string contextValue, double k, double c, double b)

{

this.contextValue = contextValue;

this.k = k;

this.c = c;

this.b = b;

}

public override double CalculateUtility(Context context)

{

double[] valueState = context.GetContextValue(contextValue);

double currentWeightedValue = CalculateWeightedValueInversed(valueState);

double utility = CalculateQuadratic(currentWeightedValue);

return CheckUtility0To1(utility);

}

private double CalculateQuadratic(double x)

{

return Math.Pow((x - c), k) + b;

}

}

}

## Codificació de la simulació

### ActionInformation

using System.Collections.Generic;

namespace Assets.Scripts

{

public class ActionInformation

{

private Dictionary<Atribute, double> specialModifiers;

private string actionName;

private double durationTimeUnits;

public double DurationSeconds

{

get { return durationTimeUnits; }

}

public string ActionName {

get { return actionName; }

}

public ActionInformation(string actionName, double durationTimeUnits)

{

this.specialModifiers = new Dictionary<Atribute, double>();

this.actionName = actionName;

this.durationTimeUnits = durationTimeUnits;

}

public Dictionary<Atribute, double> GetSpecialModifiers()

{

return specialModifiers;

}

public void AddSpecialModifier(Atribute atribute, double modifier)

{

specialModifiers.Add(atribute, modifier);

}

}

}

### Atribute

using System;

namespace Assets.Scripts

{

public class Atribute

{

private String atributeName;

private double minValue;

private double maxValue;

private double currentValue;

private double valueModifier;

public Atribute(string atributeName, double minValue, double maxValue, double valueModifier)

{

this.atributeName = atributeName;

this.minValue = minValue;

this.maxValue = maxValue;

this.currentValue = maxValue;

this.valueModifier = valueModifier;

}

public String AtributeName

{

get { return atributeName; }

}

public double MinValue

{

get { return minValue; }

}

public double MaxValue

{

get { return maxValue; }

}

public double CurrentValue

{

get { return currentValue; }

}

public double ValueModifier

{

get { return valueModifier; }

}

public void UpdateValue()

{

UpdateValue(valueModifier);

}

public void UpdateValue(double valueModifier)

{

currentValue += valueModifier;

if (currentValue > maxValue)

currentValue = maxValue;

else if (currentValue < minValue)

currentValue = minValue;

}

public Boolean TheValueIsMinimum()

{

return currentValue <= minValue;

}

public double[] GetValues()

{

double[] result = { maxValue, minValue, currentValue };

return result;

}

}

}

### GameController

using UtilityAI;

using UtilityAI.UtilityFunctionImplementations;

using Assets.Scripts;

using System.Collections.Generic;

using UnityEngine;

public class GameController : MonoBehaviour {

private const double UPDATE\_SECONDS = 1;

private const double PAUSE\_BETWEEN\_ACTIONS\_SECONDS = UPDATE\_SECONDS;

private double nextUpdate;

private double nextActionChange;

private double lastActionChange;

private GameState gameState;

private Dictionary<Action, ActionInformation> actionInformation;

private Action currentAction;

public UIController UIcontroller;

void Start ()

{

nextUpdate = Mathf.FloorToInt(Time.time) + UPDATE\_SECONDS;

actionInformation = new Dictionary<Action, ActionInformation>();

List<Atribute> atributes = BuildActionsAndAtributes();

gameState = new GameState(atributes);

PrepareActionToMake();

ShowUIInitialTexts();

UpdateMainInformation();

UpdateAtributeTexts();

}

void Update()

{

if (Time.time >= nextUpdate)

{

nextUpdate = nextUpdate + UPDATE\_SECONDS;

UpdateFixedInterval();

}

}

private void UpdateFixedInterval()

{

if (gameState.CheckCharacterDies())

{

UIcontroller.ShowDeathPanel();

this.enabled = false;

}

if (Time.time >= lastActionChange + PAUSE\_BETWEEN\_ACTIONS\_SECONDS)

{

gameState.UpdateGameState(actionInformation[currentAction].GetSpecialModifiers());

UpdateMainInformation();

UpdateAtributeTexts();

}

else

{

gameState.UpdateGameState();

UpdateMainInformationIdle();

UpdateAtributeTextsIdle();

}

if (Time.time >= nextActionChange)

PrepareActionToMake();

}

private void PrepareActionToMake()

{

List<Action> actions = new List<Action>(actionInformation.Keys);

Action bestAction = BestActionCalculator.CalculateBestAction(actions, currentAction, gameState, BestActionCalculator.BEST\_ACTION, false);

currentAction = bestAction;

lastActionChange = nextUpdate;

nextActionChange = nextUpdate + PAUSE\_BETWEEN\_ACTIONS\_SECONDS + actionInformation[currentAction].DurationSeconds;

}

private void ShowUIInitialTexts()

{

foreach (Atribute atribute in gameState.Atributes)

UIcontroller.SetAtributeInitialTexts(atribute.AtributeName, atribute.CurrentValue.ToString(),

atribute.MaxValue.ToString(), atribute.MinValue.ToString());

}

private void UpdateMainInformation()

{

string currentActionString = actionInformation[currentAction].ActionName;

UIcontroller.UpdateMainInformation(GetAverageHappiness(), GetTimeFormatted(), currentActionString);

}

private void UpdateMainInformationIdle()

{

UIcontroller.UpdateMainInformation(GetAverageHappiness(), GetTimeFormatted(), "Idle");

}

private void UpdateAtributeTexts()

{

foreach (Atribute atribute in gameState.Atributes)

{

Dictionary<Atribute, double> atributeSpecialModifiers = actionInformation[currentAction].GetSpecialModifiers();

if (atributeSpecialModifiers.ContainsKey(atribute))

UIcontroller.UpdateAtributeCurrentText(atribute.AtributeName, atribute.CurrentValue, atributeSpecialModifiers[atribute]);

else

UIcontroller.UpdateAtributeCurrentText(atribute.AtributeName, atribute.CurrentValue, atribute.ValueModifier);

}

}

private void UpdateAtributeTextsIdle()

{

foreach (Atribute atribute in gameState.Atributes)

{

UIcontroller.UpdateAtributeCurrentText(atribute.AtributeName, atribute.CurrentValue, atribute.ValueModifier);

}

}

private string GetAverageHappiness()

{

int count = 0;

double sum = 0;

foreach (Atribute atribute in gameState.Atributes)

{

sum += (atribute.CurrentValue - atribute.MinValue) / (atribute.MaxValue - atribute.MinValue);

count++;

}

int average = (int) (sum / count \* 100);

return average + "/100";

}

private string GetTimeFormatted()

{

int totalSeconds = Mathf.FloorToInt(Time.time);

int seconds = totalSeconds % 60;

int minutes = totalSeconds / 60;

if (seconds < 10)

return minutes + ":0" + seconds;

return minutes + ":" + seconds;

}

List<Atribute> BuildActionsAndAtributes()

{

Atribute fullness = new Atribute("Fullness", 0, 100, -3);

Atribute awake = new Atribute("Awake", 0, 100, -1);

Atribute relax = new Atribute("Relax", 0, 100, -1);

Atribute clean = new Atribute("Clean", 0, 100, -1);

Atribute money = new Atribute("Money", 0, 100, -2);

Atribute energy = new Atribute("Energy", 0, 100, +1);

Action action = new Action(new LogisticUtilityFunction("Fullness", 1, System.Math.E));

ActionInformation information = new ActionInformation("Eat", 4 \* UPDATE\_SECONDS);

information.AddSpecialModifier(fullness, 10);

actionInformation.Add(action, information);

action = new Action(new LogitUtilityFunction("Awake", 1, System.Math.E));

information = new ActionInformation("Sleep", 8 \* UPDATE\_SECONDS);

information.AddSpecialModifier(awake, 5);

information.AddSpecialModifier(relax, 1);

actionInformation.Add(action, information);

action = new Action(new LinearUtilityFunction("Money", 0.8, 0.2));

information = new ActionInformation("Work", 6 \* UPDATE\_SECONDS);

information.AddSpecialModifier(money, 8);

information.AddSpecialModifier(relax, -4);

actionInformation.Add(action, information);

List<UtilityFunction> composedFunctions = new List<UtilityFunction>

{

new ConstantUtilityFunction(0),

new LinearUtilityFunction("Stress", 2, -1.2)

};

double[] cutPoints = new double[] { 0.6 };

action = new Action(new ComposedUtilityFunction("Relax", composedFunctions, cutPoints));

information = new ActionInformation("Exercise", 4 \* UPDATE\_SECONDS);

information.AddSpecialModifier(relax, 15);

information.AddSpecialModifier(clean, -5);

information.AddSpecialModifier(energy, -10);

actionInformation.Add(action, information);

action = new Action(new ConstantUtilityFunction(0.3));

information = new ActionInformation("Watch TV", 2 \* UPDATE\_SECONDS);

information.AddSpecialModifier(relax, 10);

actionInformation.Add(action, information);

action = new Action(new QuadraticUtilityFunction("Clean", 5, 0, 0));

information = new ActionInformation("Shower", 3 \* UPDATE\_SECONDS);

information.AddSpecialModifier(clean, 40);

actionInformation.Add(action, information);

List<Atribute> atributes = new List<Atribute>

{

fullness,

awake,

relax,

clean,

money,

energy

};

return atributes;

}

}

### GameState

using System.Collections.Generic;

using UtilityAI;

namespace Assets.Scripts

{

public class GameState : Context

{

private List<Atribute> atributes;

public List<Atribute> Atributes

{

get { return atributes; }

}

public GameState(List<Atribute> atributes)

{

this.atributes = atributes;

}

public void UpdateGameState()

{

UpdateGameState(new Dictionary<Atribute, double>());

}

public void UpdateGameState(Dictionary<Atribute, double> specialAtributesValues)

{

foreach (Atribute atribute in atributes)

{

if (specialAtributesValues.ContainsKey(atribute))

atribute.UpdateValue(specialAtributesValues[atribute]);

else

atribute.UpdateValue();

}

}

public bool CheckCharacterDies()

{

foreach (Atribute atribute in atributes)

{

if (atribute.TheValueIsMinimum())

return true;

}

return false;

}

public double[] GetContextValue(string valueName)

{

Atribute valueAtribute = null;

foreach (Atribute atribute in atributes)

{

if (atribute.AtributeName.Equals(valueName))

{

valueAtribute = atribute;

break;

}

}

if (valueAtribute == null)

return new double[] { 1, 0, 0 };

return valueAtribute.GetValues();

}

} }

### UIAtribute

using UnityEngine;

using UnityEngine.UI;

public class UIAtribute : MonoBehaviour {

public Text atributeName;

public Text atributeCurrentVal;

public Text atributeMax;

public Text atributeMin;

public void SetInitialTexts(string atributeName, string atributeCurrentVal, string atributeMax, string atributeMin)

{

this.atributeName.text = atributeName;

this.atributeCurrentVal.text = atributeCurrentVal;

this.atributeMax.text = atributeMax;

this.atributeMin.text = atributeMin;

}

public void UpdateCurrentVal(string atributeCurrentVal)

{

this.atributeCurrentVal.text = atributeCurrentVal;

}

}

### UIController

using Assets.Scripts;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class UIController : MonoBehaviour {

public GameObject objectAtribute1;

public GameObject objectAtribute2;

public GameObject objectAtribute3;

public GameObject objectAtribute4;

public GameObject objectAtribute5;

public GameObject objectAtribute6;

public Text happiness;

public Text time;

public Text currentActivity;

public GameObject deathPanel;

private List<UIAtribute> atributes;

void Start()

{

atributes = new List<UIAtribute>

{

objectAtribute1.GetComponent<UIAtribute>(),

objectAtribute2.GetComponent<UIAtribute>(),

objectAtribute3.GetComponent<UIAtribute>(),

objectAtribute4.GetComponent<UIAtribute>(),

objectAtribute5.GetComponent<UIAtribute>(),

objectAtribute6.GetComponent<UIAtribute>()

};

deathPanel.gameObject.SetActive(false);

}

public void UpdateMainInformation(string happiness, string time, string currentActivity)

{

this.happiness.text = happiness;

this.time.text = time;

this.currentActivity.text = currentActivity;

}

public void SetAtributeInitialTexts(string atributeName, string currentValue, string maxValue, string minValue)

{

foreach (UIAtribute atribute in atributes)

if (atribute.atributeName.text.Equals("Value"))

{

atribute.SetInitialTexts(atributeName, currentValue, maxValue, minValue);

break;

}

}

public void UpdateAtributeCurrentText(string atributeName, double currentValue, double valueModifier)

{

foreach (UIAtribute atribute in atributes)

if (atribute.atributeName.text.Equals(atributeName))

if (valueModifier > 0)

atribute.UpdateCurrentVal(currentValue + " (+" + valueModifier + ")");

else

atribute.UpdateCurrentVal(currentValue + " (" + valueModifier + ")");

}

public void ShowDeathPanel()

{

deathPanel.gameObject.SetActive(true);

}

}

# Annex III - Contingut del CD

El contingut del CD és el següent:

- **JordiRomagosa TFG Framework Utility AI – Memoria**. És el document en format PDF de la memòria del treball.

- **JordiRomagosa TFG Framework Utility AI – EstudiViabilitat**. És el document en format PDF de l’estudi de viabilitat del treball.

- **JordiRomagosa TFG Framework Utility AI – Annexes**. És el document en format PDF dels annexes del treball.

- **TFG Simulation Build**. És un fitxer comprimit on s’ha inclòs el projecte compilat de Unity de la simulació, preparat per executar.

- **TFG Simulation**. És un fitxer comprimit on s’ha inclòs el projecte sencer de la simulació amb la plataforma Unity.

- **TFG\_Framework\_UtilityAI**. És un fitxer comprimit on s’ha inclòs el projecte sencer de la codificació del framework amb l’entorn de desenvolupament Microsoft Visual Studio.

- **UtilityAI.dll**. Es tracta del codi compilat del framework, en format .dll. Està preparat per importar i utilitzar en plataformes de videojocs, per si es vol testejar el seu comportament fora de la simulació realitzada en aquest treball.