Player Modeling for Role-Playing Games

Improving Bethesda's Radiant AI

Henrique Fernandes
Instituto Superior Técnico
Lisboa, Portugal
henrique.moises@tecnico.ulisboa.pt

ABSTRACT

In many other popular role-playing games, the player modeling features and overall AI are considered poor and overlooked by the AAA video game industry, despite the fact that these are proven to enhance the player experience regarding its content and narrative.

In this work, we present a Player Modeling architecture that uses a Machine Learning instance that analyses player actions and interactions with the virtual world, associates them with a player profile and creates a tailored experience that should provide better enjoyment and immersion for the player.

We do believe that such solution adds more replay value, enjoyment and a better storytelling experience to any RPG by ultimately giving the player more power to influence its gaming experience.

KEYWORDS

Player Modeling, Interactive Storytelling, Role-playing Games, Decision Tree, Player Models, Machine Learning

1 INTRODUCTION

In the scope of modern video games, especially in Role-playing Games (RPGs), player modeling is an essential feature. The creation of more immersive and engaging experiences can be achieved by making an analysis of the user and tracing his interactions with the virtual world. This analysis can then produce a player model to be used by a Drama Manager (DM) system to change some aspects of the game such as story, game content, characters actions and objectives. The use of player modeling and interactive storytelling techniques can be very beneficial to modern video games because they have the potential to craft much more player-focused experiences, improving player expected enjoyment and engagement, as well as adding replay value to the game.

The creation and use of player profiles in an interactive experience is called Player Modeling (PM) [4]. These profiles can be created through several methods and can have a large variety of applications in video games, such as adapting the game difficulty, creating personalized content and believable agents, play testing analysis and monetization of free-to-play games [31]. The use of player modeling in video games makes for a more rich and diversified experience as much of the game content, sometimes also including the story, is created according to player preferences.

Today there are some commercial systems that make use of Interactive Storytelling (IS) techniques, such as the *Radiant AI* system in Bethesda's The Elder Scrolls V: Skyrim (Bethesda 2011). This system allows the game engine to create dynamic reactions both from the virtual world and the Non-player Characters (NPCs) to the player actions. Its objective is to create a more immersive and non-linear narrative and thus a more personal, believable and unpredictable experience for the player. This system keeps track of the player actions and decisions, as well as the attributes and achievements of the character controlled by the player - player character - in order to properly adjust the game reactions to him.

2 STATE OF THE ART

Interactive Storytelling

In Interactive Storytelling (IS), the storyline is not predetermined and the narrative and its evolution can be influenced and changed by the user interactions with the experience [18]. Interactive Storytelling in RPGs can be achieved through several approaches. A simple yet largely used technique, specially in commercial systems, is to build a direct acyclic graph, also called *branching story graph* where each node contains a point in the story where the player has to make a decision (e.g. plot point) and the arcs represent the paths that the player can follow in the story.

More sophisticated implementations try to maximize the expected enjoyment of the player through several methods, such as creating player profiles [23] and trying to match the player with a predefined profile, using previous players feedback to create more appealing stories [27] or using probabilities that predict which story branch the player is more likely to follow [33].

Drama Managers

In order to keep the narrative coherent and to apply the changes made by the user to the story, a Drama Manager (DM) is often used. A DM is a background agent that monitors an interactive experience and intervenes in order to

shape the global experience so that it reflects the user's actions or choices, keeping the expressive goals by the author at the same time.

Drama Manager systems can be analyzed and categorized according to the level of autonomy that each implementation gives to the computer-controlled agents in order for the story to be created solely on the interactions between the player and these agents [9]. These systems have been divided into three types - *Centralized Drama Management*, *Distributed Drama Management* and *Mixed Drama Management*.

Centralized Drama Manager. Centralized Drama Management systems have a central entity that has full control of the actions performed by the characters and tries to react to the interactions and choices that the player takes in the virtual world, intervening and adapting the story in order to keep a coherent narrative.

Some examples include creating a player model, maximizing an experience-quality function or keeping the player on an emotional trajectory [16].

Distributed Drama Manager

Distributed Drama Management systems are based on completely autonomous agents that have the capacity to reason and choose their own actions and interactions between them and other agents or the virtual world. In this category the stories that are created completely emerge from the actions taken by the individual agents and the interactions between them, avoiding a predefined plot or a set of possible plots.

These systems give agents total autonomy to deliberate and choose their objectives and interactions with the world, other agents and the player. The plot emerges based only in these interactions - *Emergent Narrative*.

Mixed Drama Manager. In this type of systems, the agents are autonomous enough to reason and perform several interactions with the virtual world and the player that are appropriate in the context of the story.

However, a DM supervises the global plot and can order the agents to follow a certain direction in the narrative or hint the player in order to keep the development of the story, preventing the player from being stuck in a certain plot point or fall into a certain plot point that contradicts the global story. This system makes sure that certain events happen while maintaining the story structured.

Radiant AI. The video game TESV:Skyrim makes use of a DM to make the virtual world more alive and the agents more believable. This system, called *Radiant AI*, is capable of creating dynamic reactions to the changes that the player makes in the environment. More so, the agents can quickly adapt their objectives and actions in response to the player.

Overall, the system can be considered a mixed drama manager, since the NPCs actions are determined by this system when a change occurs in the game. For the most part, the NPCs have their own reasoning and follow their own objectives, like daily tasks, routines, etc. However, when certain actions are performed by the player, an event is triggered that changes the reasoning of the NPCs and makes them adapt to the new state of the world.

Player Modeling

In order for a system to adapt the game to user preferences, changing the narrative and other content, some information has to be retrieved from him. This information is obtained through an analysis of players cognitive, affective and behavioral patterns in order to create a *model* that expresses their personality, intentions and characteristics [13].

Player Modeling, from a general perspective, is the study of this data and it makes possible to perceive, record and analyze the player actions, detecting certain patterns in his interactions with the virtual world, to create a statistical model. This model is then used to create an experience that the player is more likely to enjoy and be interested in.

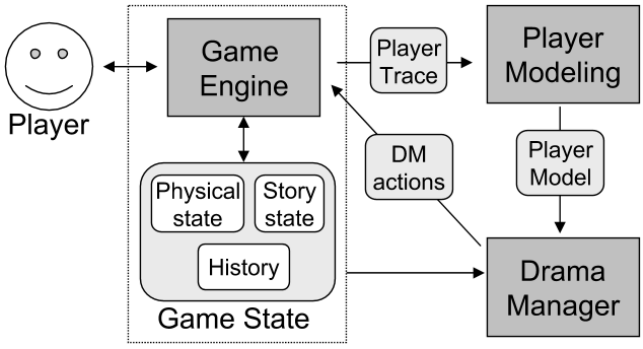

Figure 1: Player Modeling (adapted from [26]).

Player Modeling Applications. Player Modeling has a large variety of applications. The first and more important one is to adapt the game experience to the player. This includes the narrative, game content, mechanics, hints, levels, difficulty ,etc. The system captures and analyses some input given directly or indirectly by the player to decided what should be best for the player.

Other application is the creation of personalized game content. Modern days sometimes use *Procedural content generation* (PCG) [29]. This content can also include story segments or NPCs behavior and is sometimes the result of search based algorithms that try to adjust accordingly to player preferences or expected enjoyment.

The creation of believable agents is another application of player modeling [29]. Some systems make use of human

user models to change the behavior of the agents present in the game to make them feel more "natural" or less robotic.

Player Type Classifications

One of early documented division of RPG player types was formulated by Glenn Blacow [7]. The author considered that there were four aspects of adventure gaming - *Power Gaming, Role-Playing, Wargaming* and *Story Telling* - and that the interaction of those four elements created the feeling of any given adventure.

Richard Bartle published an article [3] where he postulated four different player archetypes for Multi-user Dungeons (MUDs). This classification, which was designed with a multi-player structure in mind, can also be applied to single-player games and separates players according to their preferred actions within the game. The archetypes are *Explorers, Killers, Achievers* and *Socializers*.

Another RPG player type classification was proposed by Robin D. Laws [20] that divided players into seven archetypes: *Power Gamer, Butt-Kicker, Tactician, Specialist, Method Actor, Storyteller* and *Casual Gamer*.

BrainHex. A more recent player classification, that covers all game genres and has some psychological and physiological foundations, have been postulated by video game industry consultant *International Hobo*. *BrainHex*[22] is a player classification based on behavior from seven key elements in the human nervous systems, creating the following seven classes: *Seeker, Survivor, Daredevil, Mastermind, Conqueror, Socializer* and *Achiever*.

Quest Types

There has been an increasingly interest in classifying RPG quests and missions[1], according to their content, structure or quest-giver motivations. This is specially helpful when trying to build quest generation frameworks, procedural content generation and PM implementations.

Time, Place and Objective Oriented. One video game classification consisting on three basic quest types was proposed by Aarseth [1]. The author postulates that video game quests can be divided into three main groups: *time-oriented, place-oriented* and *objective-oriented*.

NPC Motivations. Doran et al. [12] provide an extensive work on common RPG quests structures. In their work they make an analysis on the most common objectives in RPGs and MMORPGs and propose a quest classification model

[1]Quests or missions in RPGs, MMORPGs and MUDs are a set of objectives that a player or a group of players (i.e. party) may complete in order to earn a reward - be it items, money or experience - or to advance a narrative.

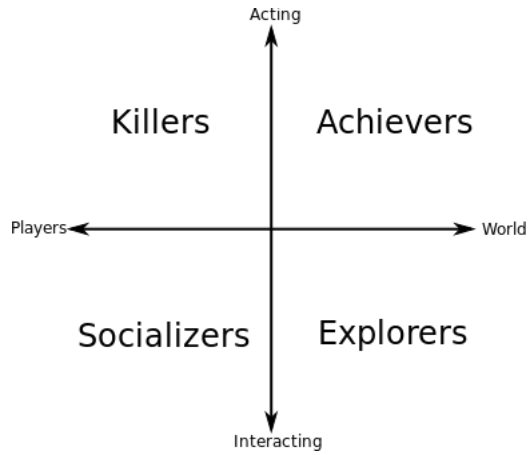

Figure 2: Bartle player taxonomy chart.

based on seven NPC motivations: Knowledge, Spirit, Comfort, Reputation, Serenity, Protection, Conquest, Wealth, Potential, Ability and Equipment.

Structure-Based. Another great analysis on MMORPG quest types is provided by Dickey et al. from the player actions perspective [10]. He used *World of Warcraft*, one of the more popular MMORPGs ever, and *ToonTown* to build a detailed mission classification according to the actions that the player character must perform in order to complete the quest or advance a small narrative, i.e. other than the main plot.

While the classification was built taking into account multi-player video games, it can be adjusted to the single-player component as well. The authors divided the quests into six groups: *Bounty, FedEx, Messenger, Collection, Escort* and *Goodwill*.

Machine Learning in Video Games

Since machine learning and the AI concept became popular, there have been a number of applications in the video game scene. The more common examples throughout the years are path-finding and decision-making by NPCs. In an effort to develop more better games, the video game industry have invested in more complex AIs and machine learning techniques. The objective varies, ranging from creating more challenging or real-like adversaries, more believable and interesting NPCs, a better overall or personalized experience from the player point-of-view and so on.

Classifying Players in MMORPGs. The use of clustering of behavior data collected during game play in online multi-player games have also been performed to model players, offering different types of players, adjusted game difficulty. Anagnostou et al [2] have implemented a CURE data

clustering algorithm [14] to classify players into two main groups - the action player and the tactical player - according to their play style. This separation allows the separation of hardcore and casual players, offering action players a more challenging experience and casual gamers a more fun, less challenging experience.

Game Persistent Agents. Merrick et al [21] presented Motivated Reinforcement Learning (MLR) agents that can explore the virtual world and evolve in response to interesting experiences in massively multi-player online video games.

In their work, the authors have implemented a MLR model in *Second Life* virtual world that allows NPCs to develop new skills according to its environment, motivations and reward signals. The learning process uses Q-learning reinforcement strategy to maximize the expected value of the total reward return over all successive steps [28].

3 PROPOSED SOLUTION

Overview

The proposed solution consists of several components which sequentially interact with each other in a series of cycles. The main components proposed are a Player Classifier Module, a Class Manager, a Content Manager and a Player Lure Module. These components interact with each other and other game elements to function properly. A description of these entities is described in this section.

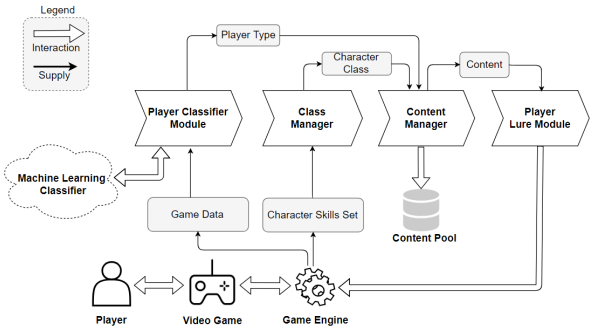

Figure 3: Proposed player modeling solution architecture.

Independent and External Components

Player. The player is the one that generates the data to the Player Classifier Module and the Class Manager, using his avatar to interact with the virtual world. It is also the player who validates the personalized content presented to him, by either accepting or rejecting it. Every action that the player performs with his character in the virtual world is a potential source of data to be used by the system. In a sense, the video game and the player character are a mere abstraction through which the player communicates with the game engine and expresses his thoughts, psychological traits, game style preference, intentions, likes and dislikes [6].

Game Engine. The game engine receives the player physical input and orchestrates all other game components to deliver the player some sort of feedback, be it visual, auditory or physical. The game engine is, in reality, the component that the player is interacting with through an abstractions such as the video game itself and the character or characters controlled by the player. More importantly, it is the game engine that contains, for a particular point in time, all the information regarding the current state of the game.

Character Skills Set. Character skills are one base premise is the RPG genre since they serve as a mean for character personalization. Generally speaking, the different types of skill progression in RPGs can be divided into three types: player-defined, game-defined and mixed.

On the first instance, once the player character have reached an experience threshold, i.e. evolved a level, the game will ask the player to assign skill and perk points to different character skills or across a skill tree; In the second instance, the game engine is in full control of the skills and perks developed by the player character and will assign them according to a certain criteria, for instance the most used skills by the character. On the mixed type, the game engine will either assign the skills or perks automatically and will let the player choose the remaining one.

The character skills set provides a second layer of personalization over the chosen content, e.g. class-related quest and dialogue. A large majority of RPGs have character classes that are defined by some attributes or uses them as guidelines for a number of features, including experience boost if the character used the class specific skills and penalties if they use others, class-limited content, NPC dialogue, etc. Most RPGs will force the player to choose a class during character creation or throughout the game, offering the possibility of changing them or not.

Player Classification. Player classification is one of the foundations of this system, since a player type is what the ML algorithm will try to obtain to select the personalized content by the Content Manager. player classification model divides players into groups according to some psychological traits, interactions with the different game elements and so on. The player characteristics used to classify a player into in a class and the number of classes vary from model to model. One player type from the selected player classification will be output by the Player Classifier Module and inputed in the Content Manager, allowing it to choose from a large set

of content, some that should maximize player enjoyment, in case the player type is correctly identified by the ML algorithm.

Content and Content Pool. Content in video games comes in many forms such as the virtual world itself, NPCs, game appearance style, narrative, mechanics, levels, difficulty level, items and so on [8]. In order to do create a tailored experience, some custom content have to be created or reused from the game to be selected by the Content Manager. It is also necessary to identify which content is suitable for which player type, e.g. a player who prefers action and a challenge may be more interested in an higher difficulty setting or a quest where he have to defeat several difficult enemies.

The Content Pool is a repository of the content that can be selected by the Content Manager. This does not mean that there must be an external database of content, but rather an identification of game features that can be used as personalized content to present the player and to what player type or types it is more suitable for.

System Components

Player Classifier Module. The Player Classifier Module is a subsystem that periodically queries the game engine for actions, events and interactions that involved the player and interacts with a ML instance in order to identify the player type that better fits a profile according to the data obtained from the game engine. Finally, it outputs the obtain player type from that interaction with a ML algorithm to the Content Manager.

Machine Learning Instance. The ML instance is the entity responsible for choosing the appropriate player type, according to the inputed values by the Player Classifier Module. The inputed values correspond to the feature values being tested in the ML instance and the output is the chosen label after classification. It is important to note that a set of relevant features must be defined *a priori*, when choosing the ML algorithm. The chosen algorithm can range from supervised options such as decision tree, Naive Bayes classification and neural networks, to more complex unsupervised learning algorithms such as clustering algorithms.

Class Manager. The Class Manager obtains the character skills set from the game engine and outputs a character to be used by the Content Manager. This module can be parallelized with the Player Classifier module if the implementation environment allows it, since it does not need the player type to operate. If the game engine already contains some built-in character class model, it can be used directly. Otherwise, a model that makes an association between skills and classes have to be built. Some subclasses can even be used to further personalize the content selection. The chosen character class is sent to the Content Manager in the end of the execution of this module.

Content Manager. This unit receives the player type output by the Player Classifier module and the character class or classes by the Class Manager module and selects fitting content to present to the player. To do this, the module queries the Content Pool on available content, using as criteria the inputed values. The delivered content can come in many forms, inclusively a sequence of instructions, e.g. change difficulty setting, send enemies to attack the player character and start quest N. Ultimately in depends only on implementation and available content. The chosen content is sent to the Player Lure Module at the end of the execution of this unit.

Player Lure Module. The Player Lure Module is an entity that receives the content selected by the Content Manager and provides instructions to the game engine on how, when and where to present the content to the player without breaking the normal game play and making it apparent that the new content came with to the original game. Let us suppose that the Content Manager selected a quest to present to the player. The Player Lure Module will instruct the game engine on how to get the player to be notified of this quest. The module could perhaps instruct the game engine to put an NPC at the end of the next dungeon that the player character enters in and engage him in conversation, giving details about the quest. Once the player knows about the quest, the Player Lure Module has fulfilled its purpose and a new player analysis can begin, independently if the player accepted the quest or not. It is important to note that the system must not disturb the usual game flow and should be able to identify the appropriate moment to order the game engine to operate.

Figure 4: The Elder Scrolls V: Skyrim gameplay.

4 IMPLEMENTATION IN TESV:SKYRIM

Overview

The proposed player modeling architecture was implemented using the *Creation Kit* and Waikato Environment for Knowledge Analysis (Weka)[2] tools. This system was developed using the available resources and considering game engine limitations.

Decision Tree Classifier and C4.5 Algorithm

The implemented Machine Learning entity from the model is composed of a decision tree created through the C4.5 algorithm [25]. The C4.5 algorithm operates using a set of training data to build a DT, grouping similar partitions of data according to a set of attributes and their *information entropy* [3]. The training set used to feed the C4.5 algorithm was composed of a series of records that associate player attributes or stats to a corresponding player type, i.e. an association between in-game actions performed by the player - attributes - and a player type - label. The purpose of the DT is to predict a player type, given a set of attributes as input.

Attributes

As the player navigates through the world of TESV:Skyrim and interacts with it, the game records his actions and keeps track of a collection of data and statistics on the player and the virtual world (e.g. Dungeons Cleared, Creatures Killed, Items Crafted, etc) in the form of global variables/counters that are available by the game engine and can be consulted inside the game, should the player want to. A total of 16 from the 90 that the engine keeps records of attributes were chosen to represent the player profile. This selection provides the player model small but diverse set of actions that differ between players and directly reflect the player behavioral states and that the player meant to perform [32].

Labels

The player types represent the labels for the decision tree classifier. In a first instance, the player types considered were the ones by Robin D. Laws[20]: *Power Gamer, Butt-Kicker, Tactician, Specialist, Method Actor, Storyteller* and *Casual Gamer*. This classifications would later be replaced by the one made by Richard Bartle' [3] taxonomy of player types, which divides player types in four archetypes: *Killers, Achievers, Explorers* and *Socializers*. The reason behind this change was the decrease in *relative absolute error* and, consequentially, *root relative error* when comparing the test results of the DTs created by the two player classification models as labels.

Classifier A, was created using the labels given by Robin D. Laws player type classification, excluding the "butt-kicker" class. Using the same attributes, a second classifier - Classifier B - was created with the labels given by Richard Bartle player type classification. The classifier A obtained a *relative absolute error* of 40.01% against 21.68% provided by classifier B when using the training set as test input for the classifier. The explanation for this phenomenon may reside in the fact that classifier B uses four different labels while classifier A uses six. This is only a problem due to the relatively reduced training set size (25 records) as it should not matter at a larger scale.

Training Data Set

The training data set used to create the classifier was composed of 25 records. Each record was obtained through a controlled play session where each player was asked to play freely for two hours and then asked to self-classify themselves using the Bartle player type classifications. This way, it was possible to collect data similar to the one the player creates while playing outside testing scenarios. The testing conditions were similar to all participants as a special save file was created specifically for this purpose. Since the objective of the experience was to map player actions to a player type or model, it was of the utmost importance that the player played according to its player subconscious model in an unbiased scenario.

Generating the Classifier

The generation of the decision trees was performed with *Weka*. A comma-separated Values (CSV) file containing the training set provided the input for the C4.5 Java implementation in *Weka*[24] - J48. The J48 algorithm ran and created the classifier, which is displayed in Figure 5 and a report, which contains a summary, detailed accuracy by class and confusion matrix. To choose the more relevant attributes in order to prune the tree, the J48 algorithm performs a MDL correction [4].

Because of the relatively small sample size, it was decided that the testing would be done using the training set as test input. The other available option were to perform cross-validation and percentage split. Those were both inviable because they need a larger set to operate correctly [19]. Using the training set as testing set, all the records will traverse the generated classifier and be classified with one of the labels.

[2] *Weka* is a machine learning software written in *Java* that contains a collection of tools and algorithms for data analysis and predictive modeling [17]. It provides an intuitive graphical user interface for data manipulation and result visualization and includes all the standard data mining tasks, i.e. data preprocessing, clustering, classification, regression, visualization, and feature selection.

[3] Information Entropy - The average amount of information produced by a stochastic source of data.

[4] Oracle Minimum Description Length (MDL) is an algorithm developed by Oracle that identifies the attributes that have the greatest influence on a target attribute, discards input fields that it regards as unimportant in predicting the target [15].

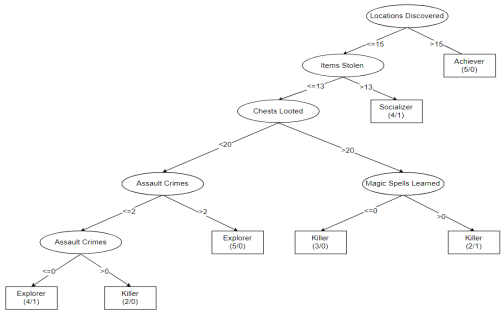

Figure 5: Decision Tree generated by the J48 algorithm execution over a Richard Bartle classification.

Resulting Decision Tree

The classifier is composed of six decision nodes and seven final leaves. The decision nodes represent a test that is made to a new instance when traversing the graph. These tests are based on the attributes that the J48 determined were the properties with more information gain [5]. Those with less information gain were pruned from the tree. The leaves represent the chosen classification for the new instance, according to the way it navigated the unidirectional graph. Because of the arguments passed on to the J48 algorithm, a leaf is only created if at least two instances of the training set end in that leaf. All the decisions are based on numeric tests due to the nature of all the attributes being numeric.

Classifier Performance Analysis

The generated report show some interesting performance indicators, making possible the evaluation of the generated DT. The considered indicators were *Precision, Recall* and *F-measure*. The three measures use the number of correctly and incorrectly classified instances for each label in different ways to better show the classifier performance.

The tests run on the Richard Bartle classifier obtained a *Precision* value of 0.890, a *Recall* value of 0.880 and a *F-measure* value of 0.876. Higher MCC[5] and ROC[6] were obtained, revealing an overall positive performance.

Decision Tree Implementation in Creation Engine

The decision tree was implemented using a *Papyrus* script. This script (`DecisionTreeQuestScript`) is attached to a sequence of states which are represented in Creation Kit as a quest (`DecisionTreeQuest`). When the initialization of this sequence of states occur, the game engine fires an event that is caught by the `OnInit` method in the script. This method displays an in-game notification stating that the DT was initialized successfully and calls the `RegisterForSingleUpdate` built-in function to mark the script for an update. This function receives as a parameter the time interval (in seconds) in which the update will be performed. The parameter inputed corresponds to 7200 seconds (two hours), which is the periodicity that we have decided to run the classifier.

There was a discussion on whether the decision tree component should be initialized immediately when starting a new game. The initial part of the unmodded version of the game is composed of a forced tutorial scenario, which is associated with the "Unbound" main story quest, to introduce new players to some game mechanics. Since the training data set data collection was performed in an unrestricted environment, there was the concern that the actions performed during the tutorial in a new game start - which are similar across every player due to the linear nature of the tutorial - could affect the player classification. It was decided that the DT component should only be initialized once the player completed the "Unbound" quest. This verification covers both scenarios where the player started a new game and completed the tutorial section; and where the player loaded a preexisting save file where he already finished the tutorial.

Decision Tree Component

When the update event is triggered, the `OnUpdate` function will run. This function is the responsible for selecting a class to the player and running the classifier, assigning him a player type. Firstly, the `DecisionTree` function is called, gathering and copying the player attributes from the last two hours, through the difference between the attributes gathered in the last classifier execution and the current ones. If this is the first classifier execution for a particular save file, then the previous attributes are assumed to be 0. This way, it is guaranteed that there is no value accumulation between classifier executions, since all these attributes are global counters stored in the game engine.

After the attributes from the two previous hours are collected, an update is preformed, assigning the current attribute values to the previous ones. An alternative way to get the current values from the previous two hours was to set the real global value to 0 after the classifier execution. However, this would break other game quests that use the real value from some of these variables.

After all the variables are collected collected, the tree algorithm is finally executed. The representation of the decision tree is represented in *Papyrus* by encapsulated *if-else* sequences, representing the test nodes of the tree. The leaves of the tree are represented by return values that correspond

[5]Matthews correlation coefficient takes into account true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes.

[6]The Receiver operating characteristic curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

to the player type. The return values are represented as integers with the following correspondence to the player types they represent: 1-Explorer; 2-Killer; 3-Achiever; 4-Socializer. In each node, the classifier will compare a certain attribute and either perform another comparison or return the player type, if it reached a leave.

Choose Class Component

The `ChooseClass` method will assign a class to the player based solely on his in-game skills. For the purpose of this work, three classes were designed: the Mage, the Hunter and the Blacksmith. Each class is a representation of a group of several skills. There are 18 in-game skills in TESV:Skyrim, so the objective was to create three classes that grouped most of the skills and represented a "theme" for the quests proposed to the player. This function computes three class weights and decide the player class that corresponds to the highest class weight. The class weights are `archeryLevel`, `blacksmithLevel` and `mageLevel` and they are computed through the average of the correspondent skills.

Quests Manager Component

The personalized content presented to the player comes in the form of custom quests. These quests were created focusing on each of the available player classes and types. A total of 18 quest structures were created, six for each player class available and were adapted from [11]. All quests were created in a way such that their starting point, their goal's location and their starting point are dynamically selected by the *RadiantAI* component. Creating such quest structure was no easy task, but this way we can guarantee that there is a larger diversity in quests and any quest can be initiated in a location that the player is near.

After the player class and type are decided, the `StartQuests` function will run, reseting all the custom quests that were initialized but not started by the player or already finished. This guarantees that, even if the player rejected a previous quest proposed to him, a similar quest can still appear in a different place in the future. The instructions inside the `StartQuests` method reset all the custom quests, making them ready to be started again, should they be chosen by the `ChooseQuest` function.

From the analysis of the most common RPG quests types and their description [11], an association was made between quest types and player types, i.e. what quest types a player from a specific type was more likely to enjoy. Because of the single-player nature of TESV:Skyrim, the *Goodwill* was removed. Furthermore, the *Collection* was segmented into two different groups: *Peacefully Collect* and *Aggressively Collect*.

The quest selection is made using as input the player class, the player type and a random factor that guarantees that the quest type is not always the same. Since there are three quest types for each player type-class combination (as described in Table 1), a probabilistic rule was implemented to confer different probability of selection for a quest type.

Quest Type /Player Type	Killer	Socializer	Explorer	Achiever
Bounty	++			+++
FedEx			++	++
Collect Peacefully		+	+++	
Collect Aggressively	+++			+
Escort	+	+++		
Messenger		++	+	

Table 1: Quest type preference per player type. The number of plus signs represent the level of interest in a quest type from a particular player type.

For each player type-class combination, there is a distribution of 50%, 35% and 15% between the three quest types that the player is progressively more likely to enjoy. An higher chance represent a stronger likelihood of the player enjoying that particular quest type according to their description, and are represented by a larger number of plus signs in the table 1.

City Lure Component

Once the system selected the appropriate quest, a city lure component is executed. This component materializes itself in the method `StartCityLure`, which is called are the end of the Choose Quest component. The purpose of this component is to gently guide the player to the chosen quest in an immersive way. By using a courier NPC, the player can be informed that someone in a certain city wants to propose a quest to the player character. Immersion wise, it is preferable to inform the player of something that is happening in game with an NPC delivering a letter than a pop-up notification appearing on the screen.

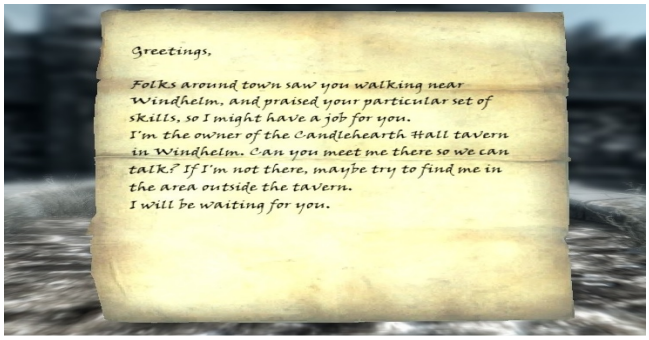

Figure 6: Example of a messenger quest letter.

The `StartCityLure` function perceives what quest was selected by the Choose Quest component and initializes the vanilla courier script, ordering the courier NPC to track the

player and deliver him a letter. The contents of the letter are defined by the type of custom quest previously selected by the system and the location of the player.

The content of the letters is personalized at the level of the location that these refer to. When the `StartCityLure` method is called, besides choosing what type of letter will be delivered to the player, the system also identifies the player character location in the game as well. Once the player character location and letter model are selected, the loose components (also known as Aliases in the Creation Engine) of the letter are filled by the Radiant AI system. These refer to the location where the player character is and, consequentially, where the quest giver is.

Custom Quests

All the quests selected by the *Quest Selector* component were created specifically for the purpose of this work. Each class has a quest for each one of the six quest types. These quests were created using the interface provided by *Creation Kit* and some *Papyrus* scripting. The way that quests relate with a class is through their plot and objectives, i.e. a quest belonging to the hunter class will have the player using his hunting skills and/or helping other hunters.

The *Creation Kit* tool provides a window just for the purpose of quest creation. The majority of the aforementioned components can be defined in the tabs of this window. More complex components such as global variables, *AI Packages* and *SM Event Nodes* must be created and edited in the appropriate sections, even if they are part of the quest.

Quest Types

In the bounty quests, the player character receives a letter indicating that there is a bounty on a creature, animal or enemy NPC, and a great rewards awaits him, should he complete the bounty.

The escort quests require the player character to escort an NPC to another town of the player choosing.

In the messenger quests, an NPC asks the player character for help to deliver a message to another NPC in any part of Skyrim.

The FedEx quest type is similar to the messenger one but requires the player to deliver, fetch or trade items with the NPCs.

Finally, the collect quests require the player character to collect a certain number of items. The difference between peaceful and aggressive collection quests reside in the fact that the aggressive one requiring the player to kill creatures or enemy NPCs to get the items, while the items asked in a peaceful quest type can be obtained in the game world without a fight.

All the quests have some similarities between them. All of them reward the player character with money or other useful items and experience in a skill belonging to the chosen class. This way, the player is encouraged to play according to his preferences and be rewarded by doing so.

Quest Example

Let us take a look at the `ArcheryMessengerQuest`, which is one of the simplest quests and part of the hunter class quests. When this quest is selected by the *Quest Selector* component, it automatically sets this stage state to 0 and runs the *City Lure* component.

The *City Lure* component will check the player character actual location and dispatch a courier with a letter. The quest giver for this particular quest is an innkeeper. Using the location of player character, the *Radiant AI* system will assign the innkeeper from the nearest town (i.e. in the same hold where the player character is) as the quest giver.

The letter will state that the innkeeper from that particular hold heard about the hunting skills of the player character and might have a mission for the player character (similar to the one present in Figure 6).

In the dialogue section of the quest creation, some dialogue branches were defined. These have the condition to only be used by the quest giver, which was the filled "Innkeeper" alias when this quest was initialized. This makes it so that only that particular innkeeper can deliver those dialogue lines. When the player reaches the inn, the innkeeper will engage in conversation with the player, explaining that he wants the player to check on a hunter in the region and get back with the news (Figure 7).

This behavior is caused by the `ForceGreet` AI package added to all the innkeepers in the game. This AI package orders the NPC to deliver a predefined dialogue line when the stage for this particular quest is set to 0. It is also important to note that the innkeepers will continue to behave like they would in an unaltered version of the game.

Figure 7: Example of a quest narrative.

If the player accepts the quest, the quest stage will be set to 10. There is an objective defined for this stage, which points to the "Hunter" alias, which was filled by the *RadiantAI* with

a reference to a random hunter in the region that the player character was in when the quest was initialized. At this point, the objective "Check on the hunter" is displayed inside the game quest list and as a marker in the compass and map.

When the player finds the hunter and talks to him, he tells the player that everything is fine with him and he have been busy hunting. After that, the quest stage is changed to 20. Again, all the dialogue, dialogue conditions and post-dialogue behavior was specified in the dialogue tab. A new objective is now displayed, since it is the objective associated with stage 20. This objective points to the "Innkeeper" alias, which was the quest giver. The new objective "Return to with the news to the innkeeper" is now displayed in the quest list and as a maker on the game map and compass.

When the player character returns to the innkeeper and tells him the whereabouts of the hunter, he will reward him with some gold coins and the player will gain experience in the archery skill (as this was a hunting-related quest). After the player character receives the rewards, the quest stage is set to 40, its finishing stage.

5 SOLUTION EVALUATION

The results of this work were evaluated both quantitatively and qualitatively. In this chapter we make a description and analysis of the methodologies and obtained results, in order to understand how the solution performed and what was the general feedback regarding it.

Testing Group Evaluation

A test was conducted to validate the used player classification and the implemented solution. It was performed in a testing group composed 30 subjects. The objective of this test was to assign the players the correct and "opposed" player type during two separate gameplay instances to understand if the difference in content was noticeable by the subject, which session was preferred and how was it perceived. Before the play sessions each participant was asked to take the Richard Bartle test, which provided us the correct player class.

This test was done in a special environment, with prepared save files for each player type. At the beginning of each play session, the correct or incorrect save file was loaded and the subject was asked to play freely for 1 hour. After each gameplay session, the subject was asked to answer one of a two-parts survey, containing questions about how he felt during and after the playing session. The conducted test followed the following protocol:

(1) **Bartle Test** - The participant is asked to take the Bartle test. The result of this test gives us the player class that we nominate as the correct one.
(2) **First Play Session** - A save file is chosen for the player. This could be the correct or incorrect version, i.e. the save file that contains a character with the same player type that the participant obtained in the Bartle test or the opposite one. The participant is then asked to play the game freely for one hour.
(3) **Questionnaire Part One** - The participant is asked to fill the corresponding part of the questionnaire in relevance to how he felt during and after the gaming experience.
(4) **Second Play Session** - Another save file is loaded. If the correct version was selected in the first play session, the incorrect version is now loaded and *vice-versa*. The subject is again asked to play for one hour.
(5) **Questionnaire Part Two** - The participant is asked to fill the corresponding part of the questionnaire in relevance to how he felt during and after his last gaming experience.

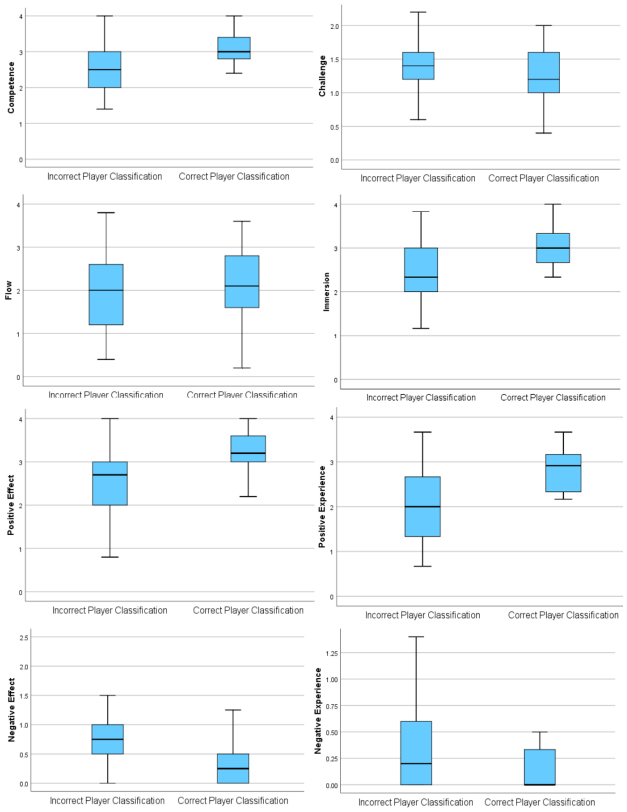

Figure 8: GEQ areas score comparison.

Results

In this section an analysis is performed on the obtained results from the questionnaire performed by each test subject. This will validate the used player classification and overall experience provided by the added content to the game. The

	Incorrect Quests for Player Type			Correct Quests for Player Type			
	Mean	Median	Standard Deviation	Mean	Median	Standard Deviation	Significance (p-value)
Competence	2.427	2.500	0.737	**3.027**	3.000	0.475	0.0001
Immersion	2.461	2.333	0.655	**2.989**	3.000	0.501	0.001
Flow	1.927	2.000	0.870	2.167	2.100	0.840	0.193
Annoyance	0.622	0.667	0.516	**0.111**	0.000	0.307	0.00004
Challenge	1.413	1.400	0.407	1.240	1.200	0.438	0.084
Negative Effect	0.858	0.750	0.579	**0.305**	0.250	0.357	0.0001
Positive Effect	2.560	2.700	0.711	**3.180**	3.200	0.462	0.0001
Positive Experience	1.967	2.000	0.863	**2.77**2	2.917	0.561	0.0001
Negative Experience	0.360	0.200	0.412	**0.139**	0.000	0.196	0.022

Table 2: Comparison of the questionnaire answers per GEQ area.

results cover all the areas assessed by the (Game Experience Questionnaire) GEQ test. However, the ones that we were more interested in were Immersion, Flow and Positive Effect during gameplay and Positive Experience post-gameplay.

The statistic metrics used are composed of the mean, median and standard deviation of scores obtain from the participants in the different GEQ areas. Due to the nature of the tests, it was also possible to perform a Wilcoxon signed-rank test [30] to understand the levels of significance between the results provided by the two classifications, given by a *p-value*. A side-by-side comparison allows as to draw some conclusions. A summary of the mean, median, standard deviation for each GEQ area and the significance between the results can be seen in Table 2.

From a general analysis, it is clear that the participants preferred the correct player classification over an incorrect one. Looking at the Table 2 the mean values for the Competence, Immersion, Flow, Positive Effect and Positive Experience areas for the correct player classification are higher than its counterpart. More over, the values in the areas of Annoyance, Challenge, Negative Effect and Negative experience are higher when the participants played with the incorrect classification save file.

Community Mod Feedback and Survey Results

To assess the TESV:Skyrim player community, the "Your Own Skyrim" game modification was published in the *NexusMods* modding content website and the digital distribution platform *Steam*, in the *Steam Workshop*.

On both *NexusMods* and *Steam Workshop* the "Your Own Skyrim" mod exceeded the expectations. The feedback and reception by the player community was very gratifying, with over 10 thousand mod downloads as of the beginning of May 2018 and a high level of interest on the issues addressed in this work.

Figure 9: "Your Own Skyrim" in the top weekly and monthly mods from the TESV:Skyrim *Steam Workshop*.

NexusMods. The game modification was published in the *NexusMods* website in the beginning of April 2018 and quickly captured the interest of many. Over the two next weeks after being published it was in the top 10 most popular Skyrim Mods of the week and top 10 trending Skyrim mods, reaching the 16th place in the most popular mods of April 2018. As of May 2018, 2581 downloads were performed on "Your Own Skyrim" mod, a community member offered his help to translate the mod to Spanish and a mod developer asked permission to use mod in 3 very popular mods that he was working on.

Steam Workshop. On *Steam Workshop* the results were even better. The "Your Own Skyrim" mod was in the top 10 Skyrim Mods of the week for the two following weeks after being published, reaching the first place during the second week for the top Skyrim mods of the week and top Skyrim mods of the month with a five star rating, as seen in Figure 9. As of May 2018 the total number of downloads for the mod from the *Workshop* was 7609.

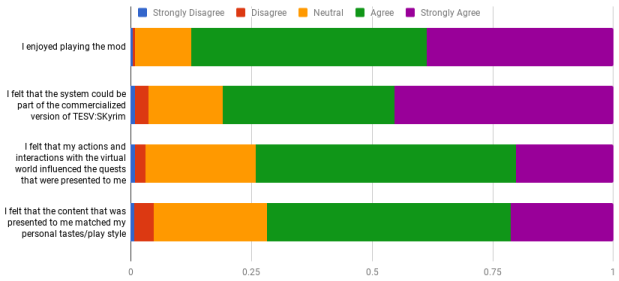

Figure 10: Online survey on YOS mod summary.

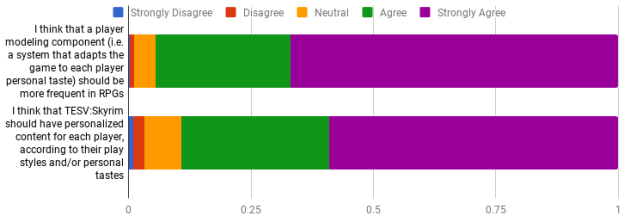

Figure 11: Summary of some answers from the mod evaluation online survey.

Player Feedback

A total of 447 people answered the questionnaire provided in the mod description. The questionnaire contained 9 questions that aimed to assess the opinion of those who downloaded the mod regarding the mod itself and PM systems in modern RPGs. Let us make an analysis on the obtained results. The population that answered the survey is composed of players that play an average of 2 to 10 hours per week and have also played TESV:Skyrim form a combined total of more than 200 hours. Regarding the use of mods, 39,4% of the participants usually play with more than 100 mods installed. As for the mod itself, 38,7% of the surveyed played it for more than 4 hours, 22,6% between 2 to 4 hours and 38,7% played it for 2 or less hours.

The last two sets of questionnaire questions are performed using a series of statements for the surveyed to answer in a 5 point Likert scale, with 1 being "Strongly Disagree" and 5 being "Strongly Agree". The first one assess the players opinion and experience with the mod. A summary of some of the given responses is displayed in Figure 10.

At the end of the questionnaire, a set of three questions rates the respondents thoughts on PM and its applications on RPGs in general. The results show a great interest in this type of PM applications, as seen in Figure 11.

6 CONCLUSIONS

The Elder Scrolls V: Skyrim was the chosen platform to develop a solution implementation because of its open-world role-playing nature, its unquestionable success and popularity, and the accessibility to game modification tools - the *Creation Kit*. The Richard Bartle player classification was the one used for this particular implementation but other player models were also considered.

As for the player type assignment algorithm, a decision tree was used, which was generated using the C4.5 decision tree generation algorithm over a training set composed of 25 records. The content presented to the player according to his classification was developed as a large variety of in-game quests which have some elements, such as objectives, locations and quest givers, that are generated during runtime with a game AI system - *Radiant AI*.

The developed solution was released to the public as a game modification - the "Your Own Skyrim - Decision Tree Classifier" mod - which was subject to appreciation by a selected group of players that formed a testing group and the game community.

From the results of the tests conducted with the testing group it was concluded that the chosen player classification model is appropriate and that there was a significant increase in player enjoyment and imaginative immersion when the players were assigned a save file with the correct player type versus an incorrect one. From the game community questionnaire it was concluded there was high approval rate, with 87,2% of those surveyed responding that they had enjoyed the mod, 74,4% agreeing that presented content was determined by their in-game actions and 71,6% concurring that it matched their preferences an play style.

Furthermore, it was also concluded that there is a high demand for player modeling systems in modern role-playing video games, with an impressive 94,6% of the respondents agreeing that PM components should be more frequent in RPGs.

The outcome of this work and the obtained results are indeed very promising. We managed to develop a solid PM architecture based on a valid classification model and using a ML technique that can be replicated in commercial game versions. The outcome of the conducted tests on the final solution allows us to support the hypothesis that this type of systems should be invested on during game development by the AAA video game industry in order to create more immersive and believable virtual worlds, maximize enjoyment and an overall better experience for the player, thus adding engagement and replay value to the video game.

REFERENCES

[1] Espen Aarseth. 2003. Playing Research: Methodological approaches to game analysis. In *Proceedings of the digital arts and culture conference.*

28–29.

[2] Kostas Anagnostou and Manolis Maragoudakis. 2009. Data mining for player modeling in videogames. In *Informatics, 2009. PCI'09. 13th Panhellenic Conference on*. IEEE, 30–34.

[3] Richard Bartle. 1996. Hearts, clubs, diamonds, spades: Players who suit MUDs. *Journal of MUD research* 1, 1 (1996), 19.

[4] Carole Beal, Joseph Beck, David Westbrook, Marc Atkin, and Paul Cohen. 2002. Intelligent modeling of the user in interactive entertainment. In *AAAI Stanford Spring Symposium, Stanford CA*.

[5] Neeraj Bhargava, Girja Sharma, Ritu Bhargava, and Manish Mathuria. 2013. Decision tree analysis on j48 algorithm for data mining. *Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering* 3, 6 (2013).

[6] Staffan Bjork and Jussi Holopainen. 2004. Patterns in game design (game development series). (2004).

[7] Glen Blacow. 1980. Aspects of adventure gaming. *Different worlds* 10 (1980), 36–39.

[8] Jeffrey E Brand, Scott Knight, and Jakub Majewski. 2003. The Diverse Worlds of Computer Games: A Content Analysis of Spaces, Populations, Styles and Narratives.. In *DiGRA Conference*.

[9] Chris Broekema. 2013. Drama Management Implementation for Interactive Digital Storytelling. (2013).

[10] Michele D Dickey. 2007. Game design and learning: A conjectural analysis of how massively multiple online role-playing games (MMORPGs) foster intrinsic motivation. *Educational Technology Research and Development* 55, 3 (2007), 253–273.

[11] Michele D Dickey. 2015. *Aesthetics and design for game-based learning*. Routledge.

[12] Jonathon Doran and Ian Parberry. 2010. Towards procedural quest generation: A structural analysis of RPG quests. *Dept. Comput. Sci. Eng., Univ. North Texas, Tech. Rep. LARC-2010-02* (2010).

[13] Anders Drachen, Alessandro Canossa, and Georgios N Yannakakis. 2009. Player modeling using self-organization in Tomb Raider: Underworld. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, 1–8.

[14] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. 1998. CURE: an efficient clustering algorithm for large databases. In *ACM Sigmod Record*, Vol. 27. ACM, 73–84.

[15] Mark H Hansen and Bin Yu. 2001. Model selection and the principle of minimum description length. *J. Amer. Statist. Assoc.* 96, 454 (2001), 746–774.

[16] Sergio Poo Hernandez, Vadim Bulitko, and Marcia Spetch. 2015. Keeping the Player on an Emotional Trajectory in Interactive Storytelling. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

[17] Geoffrey Holmes, Andrew Donkin, and Ian H Witten. 1994. Weka: A machine learning workbench. In *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*. IEEE, 357–361.

[18] Margaret Thomas Kelso, Peter Weyhrauch, and Joseph Bates. 1993. Dramatic presence. *PRESENCE: Teleoperators & Virtual Environments* 2, 1 (1993), 1–15.

[19] Ron Kohavi et al. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, Vol. 14. Montreal, Canada, 1137–1145.

[20] Robin D Laws. 2002. *Robin's laws of good game mastering*. Steve Jackson Games.

[21] Kathryn Merrick and Mary Lou Maher. 2006. Motivated reinforcement learning for non-player characters in persistent computer game worlds. In *Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*. ACM, 3.

[22] Lennart E Nacke, Chris Bateman, and Regan L Mandryk. 2014. BrainHex: A neurobiological gamer typology survey. *Entertainment computing* 5, 1 (2014), 55–62.

[23] Mark J Nelson and Michael Mateas. 2008. Another look at search-based drama management. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*. International Foundation for Autonomous Agents and Multiagent Systems, 1293–1298.

[24] Tina R Patil and SS Sherekar. 2013. Performance analysis of Naive Bayes and J48 classification algorithm for data classification. *International Journal of Computer Science and Applications* 6, 2 (2013), 256–261.

[25] J Ross Quinlan. 2014. *C4. 5: programs for machine learning*. Elsevier.

[26] Manu Sharma, Santiago Ontañón, Manish Mehta, and Ashwin Ram. 2010. Drama management and player modeling for interactive fiction games. *Computational Intelligence* 26, 2 (2010), 183–211.

[27] Manu Sharma, Santiago Ontanón, Christina R Strong, Manish Mehta, and Ashwin Ram. 2007. Towards Player Preference Modeling for Drama Management in Interactive Stories.. In *FLAIRS Conference*. 571–576.

[28] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

[29] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. 2011. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 172–186.

[30] Frank Wilcoxon, SK Katti, and Roberta A Wilcox. 1970. Critical values and probability levels for the Wilcoxon rank sum test and the Wilcoxon signed rank test. *Selected tables in mathematical statistics* 1 (1970), 171–259.

[31] Georgios N Yannakakis, Pieter Spronck, Daniele Loiacono, and Elisabeth André. 2013. Player modeling. *Dagstuhl Follow-Ups* 6 (2013).

[32] Georgios N. Yannakakis, Pieter Spronck, Daniele Loiacono, and Elisabeth André. 2013. Player Modeling. In *Artificial and Computational Intelligence in Games*, Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius (Eds.). Dagstuhl Follow-Ups, Vol. 6. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 45–59. https://doi.org/10.4230/DFU.Vol6.12191.45

[33] Hong Yu and M Riedl. 2015. Optimizing Players' Expected Enjoyment in Interactive Stories. In *Proceedings of the 11th Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.