BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

MACHINE LEARNING ALGORITHMS FOR DYNAMIC DIFFICULTY IN GAMES

ESSAY

AUTHOR **Ing. OLENA PASTUSHENKO**
AUTOR PRÁCE

BRNO 2018

Contents

Chapter 1

Introduction

In every job that must be done, there is an element of fun.
You find the fun, and - SNAP - the job's a game!

Mary Poppins

Training and development has been approached in many ways to deal with specific learning objectives. From physical training manuals, to training videos and eLearning units, every medium has its strengths and weaknesses. The lynchpin that determines the success of any training effort is engagement. Increasing students' motivation is an essential task during the educational process. One of the possible ways how to achieve this is to use innovative educational mechanics, such as gamification or serious games. Gamification provides an opportunity to extend regular learning management systems and virtual learning environments with game-like elements, such as points, levels, and meaningful narrative. With a serious game initiative, the learning content is delivered in a game-based environment.

To make these approaches even more personalized and effective, Machine Learning (ML) usage might be considered. General information about ML, main tasks and real-life examples are in Chapter 2. This chapter is mainly based on the information from the book „Designing machine learning systems with Python“ [6].

The next Chapter 3 describes one of the possible ML applications - dynamic difficulty adjustment. To make the game (or a gamified assignment) interesting for a player it is important to keep their attention and engagement in a state of flow. In order to achieve this, tasks should have increasing difficulty, adjusting to the new mastered skills of the user. Since all users have different learning curves, static difficulty changes will not work for everyone. That's why ML are useful here in order to classify users skills level and adjust the difficulty accordingly. A lot of this chapter is based on a research in [9], which studies general theory around dynamic difficulty, without particular application to gamified educational assignments.

Chapter 2

Machine learning

Machine Learning (ML) is a branch of computer science where you build models using available data. ML systems are widely used in various areas of our nowadays lives, such as: health-care, applied physics, nutrition research, economic modeling, analytics [6].

A *task* is a specific activity conducted over a period of time. There are human tasks, such as planning, designing and implementing. And also machine tasks, such as classification, regression and etc). And one of the goals of designing ML systems is to delegate as much tasks as possible to a machine. It might be challenging to match real problems to ML tasks.

Machine learning tasks occur in three broad settings:

- *Supervised learning*: The goal here is to learn a model from labeled training data that allows predictions to be made on unseen future data.
- *Unsupervised learning*: Here we deal with unlabeled data and our goal is to find hidden patterns in this data to extract meaningful information.
- *Reinforcement learning*: The goal here is to develop a system that improves its performance based on the interactions it has with its environment. This usually involves a reward signal. This is similar to supervised learning, except that rather than having a labeled training set, reinforcement learning uses a reward function to continually improve its performance.

2.1 Real-life examples of machine learning application

Early famous ML examples include, for example, the Mr Clippy office assistant, who was included in the early versions of the Microsoft office suite. That one is considered to be not a successful ML algorithm, because a lot of users found it annoying. Though, it was more of a design failure, that an algorithm mistake.

The next list shows just some of the enormous possibilities of ML applications.

- **Google Search Engine.**[1] In 2015, Google introduced RankBrain – a machine learning algorithm used to decipher the semantic content of a search query. Through the use of an intuitive neural network, RankBrain identifies the intent behind a user's search and offers them tailored information on that particular topic. RankBrain now handles around 15 percent of Google's daily queries, working out the intent behind never before seen searches much faster than the previous old rules-based system.

[1] https://www.google.com

- **Siri.** Voice recognition systems such as Siri[2] and Cortana[3] use machine learning and deep neural networks to imitate human interaction. As they progress, these apps will learn to 'understand' the nuances and semantics of our language (Fig. 2.1). For example, Siri can identify the trigger phrase 'Hey Siri' under almost any condition through the use of probability distributions. By selecting appropriate speech segments from a recorded database, the software can then choose responses that closely resemble real-life conversation.

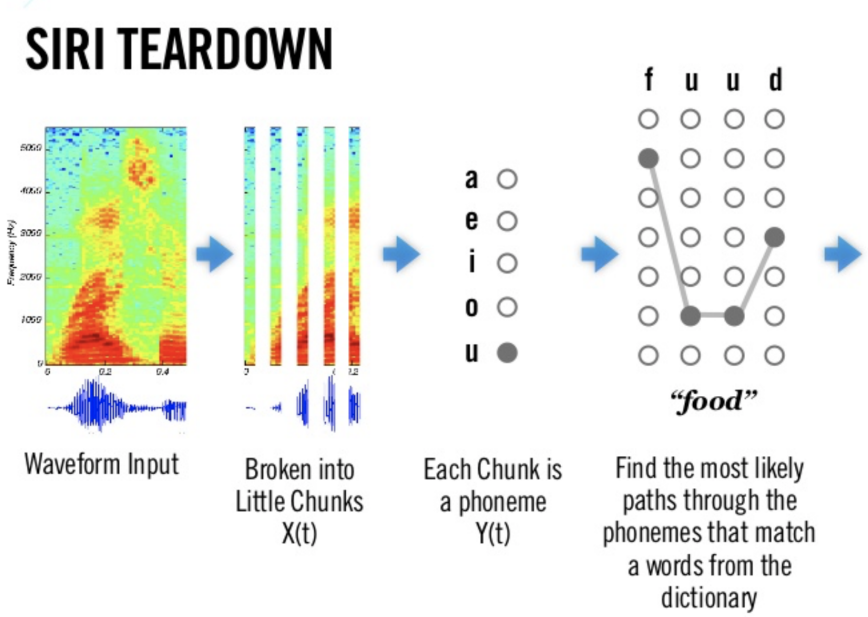

Figure 2.1: Explanation of how Siri voice decoder works. Image source: https://www.slideshare.net/shilman/ignite-seoul-machine-learning

.

- **Uber.**[4] ML is a fundamental part of the Uber model[5]. The tech giant uses these algorithms to determine arrival times, pick-up locations, and UberEATS' delivery estimations. When you book a car, Uber's aim is to estimate its arrival time as accurately as possible. Machine learning enables it to do this by analysing data from millions of previous trips and applying it to your specific situation. The same goes for UberEATS, which takes thing such as food preparation time into account to give you the best possible prediction of delivery time. The real-time analysis of these datasets has improved Uber's estimations by 26 percent and increased customer satisfaction in the process.

- **PayPal** uses machine learning algorithms to detect and combat fraud. By implementing deep learning techniques, PayPal can analyse vast quantities of customer data and evaluate risk in a far more efficient manner. Traditionally, fraud detection algorithms have dealt with very linear results: fraud either has or hasn't occurred. But with machine learning and neural networks, PayPal is able to draw upon financial, machine,

[2]https://www.apple.com/ios/siri/
[3]https://www.microsoft.com/en-us/cortana
[4]http://uber.com
[5]https://eng.uber.com/michelangelo/

and network information to provide a deeper understanding of a customer's activity and motives.

- **Spotify**[6] uses machine learning to figure out users likes and dislikes and provides them with a list of related tracks. In its Discover Weekly promotion, Spotify rounds up 30 tracks it thinks a user should listen to and delivers them in one easy-to-navigate playlist. These songs are all 'hand-picked' by machine learning algorithms, which analyse user activity and match their tastes to music with similar meta-tags.

- **Data Mining.** Just to give a brief overview of ML possibilities in Data Mining, I'd like to describe the research [1] which studied the expression of emotion in 20th century books. With access to a large volume of digitized text through the project Gutenberg digital library, WordNet[7], and Google's Ngram database [8], the authors of this study were able to map cultural change over the 20th century as reflected in the literature of the time. They did this by mapping trends in the usage of the mood words. The results are quite interesting. For example, Figure 2.2 shows the joy-sadness score for books written in the period of 1900-2000 years, and it clearly shows a negative trend associated with the period of World War II.

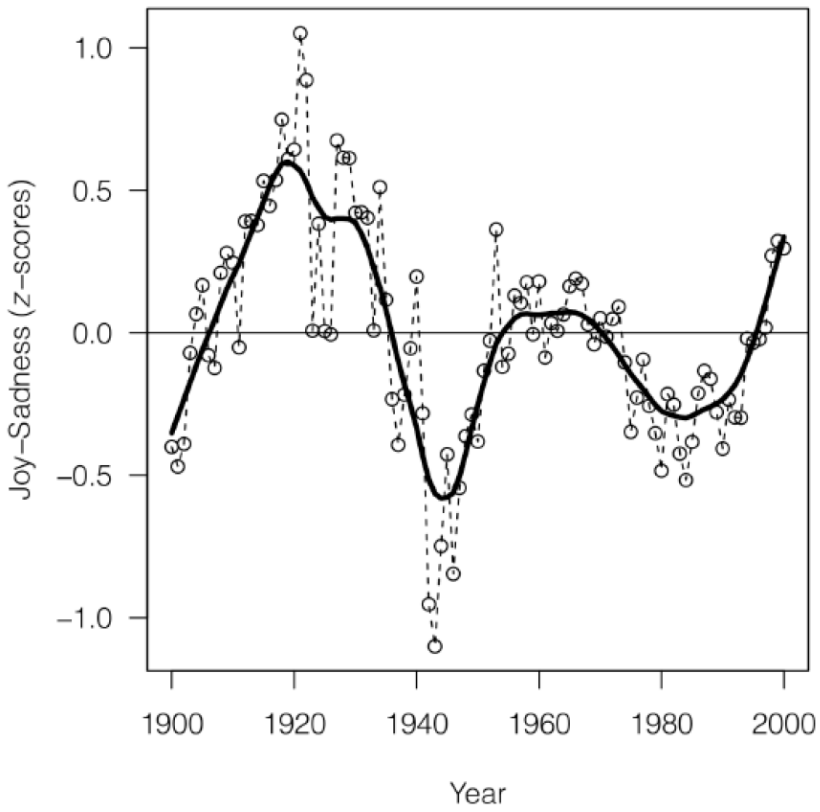

Figure 2.2: Joy-Sadness expressions ratio retrieved from the books written in 1900-2000 years. Image source: [1]

[6] https://www.spotify.com/us/
[7] http://wordnet.princeton.edu/wordnet/
[8] books.google.com/ngrams

2.2 Classification

Major machine learning tasks can be divided into several groups (Fig. 2.3).

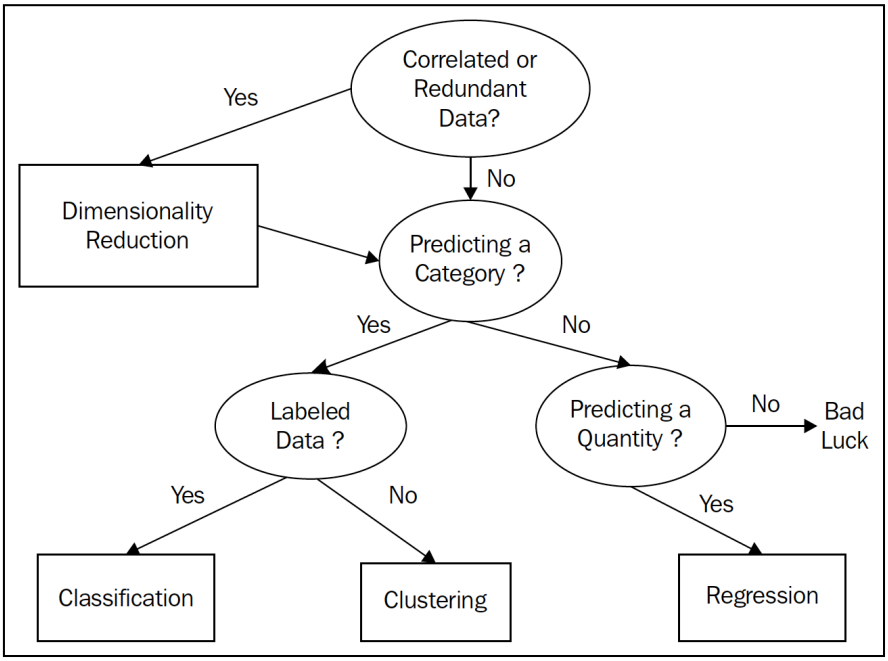

Figure 2.3: What type of task is appropriate for different machine learning problems[6].

.

Classification is probably the most common type of task; this is due in part to the fact that it is relatively easy, well understood, and solves a lot of common problems.Classification is about assigning classes to a set of instances, based on their features. This is a supervised learning method because it relies on a labeled training set to learn a set of model parameters. This model can then be applied to unlabeled data to make a prediction on what class each instance belongs to. There are broadly two types of classification tasks: binary classification and multiclass classification. A typical binary classification task is e-mail spam detection. Here we use the contents of an e-mail to determine if it belongs to one of the two classes: spam or not spam.

An example of multiclass classification is handwriting recognition, where we try to predict a class, for example, the letter name. In this case, we have one class for each of the alpha numeric characters. Multiclass classification can sometimes be achieved by chaining binary classification tasks together, however, we lose information this way, and we are unable to define a single decision boundary. For this reason, multiclass classification is often treated separately from binary classification.

In this research Classification would be investigated and discussed with more details, because this task is going to be needed for further steps. The information in this section is mainly based on the research [8].

Supervised classification is one of the tasks most frequently carried out by so-called Intelligent Systems. Thus, a large number of techniques have been developed based on Artificial Intelligence (Logical/Symbolic techniques), Perceptron-based techniques and Statistics (Bayesian Networks, Instance-based techniques).

Inductive machine learning is the process of learning a set of rules from instances (examples in a training set), or more generally speaking, creating a classifier that can be used to generalize from new instances. The process of applying supervised ML to a real-world problem is described in Figure 2.4.

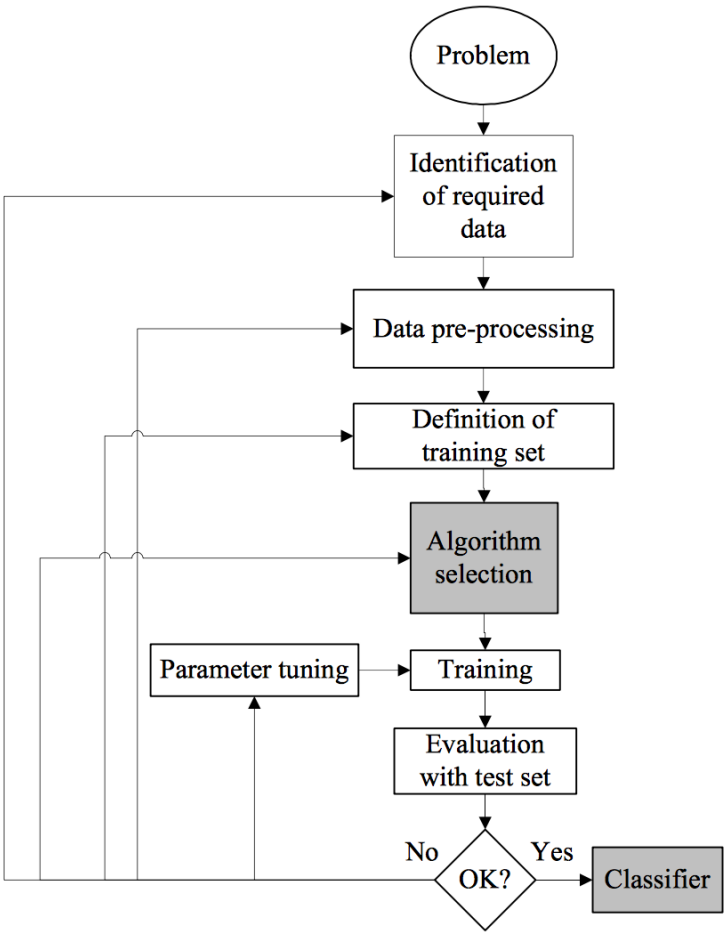

Figure 2.4: The process of supervised ML.

The first step is collecting the dataset. If a requisite expert is available, then s/he could suggest which fields (attributes, features) are the most informative. If not, then the simplest method is that of "brute-force," which means measuring everything available in the hope that the right (informative, relevant) features can be isolated. However, a dataset collected by the "brute-force" method is not directly suitable for induction. It contains in most cases noise and missing feature values, and therefore requires significant pre-processing.

The second step is the data preparation and data pre- processiong. Depending on the circumstances, researchers have a number of methods to choose from to handle missing data. These researchers have identified the techniques' advantages and disadvantages. Instance selection is not only used to handle noise but to cope with the infeasibility of learning from very large datasets. Instance selection in these datasets is an optimization problem that attempts to maintain the mining quality while minimizing the sample size. It reduces data and enables a data mining algorithm to function and work effectively with very large datasets. There is a variety of procedures for sampling instances from a large dataset.

Feature subset selection is the process of identifying and removing as many irrelevant and redundant features as possible. This reduces the dimensionality of the data and enables data mining algorithms to operate faster and more effectively. The fact that many features depend on one another often unduly influences the accuracy of supervised ML classification models. This problem can be addressed by constructing new features from the basic feature set. This technique is called feature construction/transformation. These newly generated features may lead to the creation of more concise and accurate classifiers. In addition, the discovery of meaningful features contributes to better comprehensibility of the produced classifier, and a better understanding of the learned concept.

2.3 Regression

There are cases where what we are interested in are not discrete classes, but a continuous variable, for instance, a probability. These types of problems are regression problems. The aim of regression analysis is to understand how changes to the input, independent variables, effect changes to the dependent variable. The simplest regression problems are linear and involve fitting a straight line to a set of data in order to make a prediction. This is usually done by minimizing the sum of squared errors in each instance in the training set. Typical regression problems include estimating the likelihood of a disease given a range and severity of symptoms, or predicting test scores given past performance.

2.4 Clustering

Clustering is the most well known unsupervised method. Here, we are concerned with making a measurement of similarity between instances in an unlabeled dataset. We often use geometric models to determine the distance between instances, based on their feature values. We can use an arbitrary measurement of closeness to determine what cluster each instance belongs to. Clustering is often used in data mining and exploratory data analysis. There are a large variety of methods and algorithms that perform this task, and some of the approaches include the distance based method, as well as finding a center point for each cluster, or using statistical techniques based on distributions.

2.5 Dimensionality reduction

Many data sets contain a large number of features or measurements associated with each instance. This can present a challenge in terms of computational power and memory allocation. Also many features may contain redundant information or information that is correlated to other features. In these cases, the performance of our learning model may be significantly degraded.

Dimensionality reduction is most often used in feature prepossessing; it compresses the data into a lower dimension sub space while retaining useful information. Dimensionality reduction is also used when we want to visualize data, typically by projecting higher dimensions onto one, two, or three dimensions.

2.6 Errors

In machine learning systems, software flaws can have very serious real world consequences; what happens if an algorithm, embedded in an assembly line robot, classifies a human as a production component? Clearly, in critical systems, it is needed to plan for failure. There should be a robust fault and error detection procedure embedded in the design process and systems.

Sometimes it is necessary to design very complex systems simply for the purpose of debugging and checking for logic flaws. It may be necessary to generate data sets with specific statistical structures, or create artificial humans to mimic an interface. For example, developing a methodology to verify that the logic of the design is sound at the data, model, and task levels.

Errors can be hard to track, but is always needed to assume that there are errors in the system and try to prove otherwise. It is needed to be able to capture, in the models, the ability to learn from an error. Consideration must be given to how we select our test set, and in particular, how representative it is of the rest of the dataset. For instance, if it is noisy compared to the training set, it will give poor results on the test set, suggesting that the model is overfitting, when in fact, this is not the case. To avoid this, a process of cross validation is used. This works by randomly dividing the data into, for example, ten chunks of equal size. We use nine chunks for training the model and one for testing. We do this 10 times, using each chunk once for testing. Finally, we take an average of test set performance. Cross validation is used with other supervised learning problems besides classification, but unsupervised learning problems need to be evaluated differently.

With an unsupervised task we do not have a labeled training set. Evaluation can therefore be a little tricky since we do not know what a correct answer looks like. In a clustering problem, for instance, we can compare the quality of different models by measures such as the ratio of cluster diameter compared to the distance between clusters. However, in problems of any complexity, we can never tell if there is another model, not yet built, which is better.

2.7 Unified modeling language

Machine learning systems can be complex. It is often difficult for a human brain to understand all the interactions of a complete system. We need some way to abstract the system into a set of discrete functional components. This enables us to visualize our system's structure and behavior with diagrams and plots.

Unified modeling language (UML) is a formalism that allows us to visualize and communicate our design ideas in a precise way. We implement our systems in code, and the underlying principles are expressed in mathematics, but there is a third aspect, which is, in a sense, perpendicular to these, and that is a visual representation of our system. The process of drawing out your design helps conceptualize it from a different perspective. Perhaps we could consider trying to triangulate a solution.

Conceptual models are theoretical devices for describing elements of a problem. They can help us clarify assumptions, prove certain properties, and give us a fundamental understanding of the structures and interactions of systems. UML arose out of the need to both simplify this complexity and allow our designs to be communicated clearly and unambiguously to team members, clients, and other stakeholders. A model is a simplified representation of a real system. Here, we use the word model in a more general sense,

as compared to its more precise machine learning definition. UML can be used to model almost any system imaginable. The core idea is to strip away any irrelevant and potentially confusing elements with a clear representation of core attributes and functions.

2.7.1 Class diagrams

The class diagram models the static structure of a system. Classes represent abstract entities with common characteristics. They are useful because they express, and enforce, an object-oriented approach to our programming. We can see that by separating distinct objects in our code, we can work more clearly on each object as a self-contained unit. We can define it with a specific set of characteristics, and define how it relates to other objects. This enables complex programs to be broken down into separate functional components. It also allows us to subclass objects via inheritance. This is extremely useful and mirrors how we model the particularly hierarchical aspect of our world (that is, programmer is a subclass of human, and Python programmer is a subclass of programmer).

Object programming can speed up the overall development time because it allows the reuse of components. There is a rich class library of developed components to draw upon. Also, the code produced tends to be easier to maintain because we can replace or change classes and are able to (usually) understand how this will affect the overall system.

In truth, object coding does tend to result in a larger code base, and this can mean that programs will be slower to run. In the end, it is not an „either, or“ situation. For many simple tasks, you probably do not want to spend the time creating a class if you may never use it again. In general, if you find yourself typing the same bits of code, or creating the same type of data structures, it is probably a good idea to create a class.

The big advantage of object programming is that we can encapsulate the data and the functions that operate on the data in one object. These software objects can correspond in quite a direct way with real world objects. Designing object-oriented systems may take some time, initially. However, while establishing a workable class structure and class definitions, the coding tasks required to implement the class becomes clearer.

Creating a class structure can be a very useful way to begin modeling a system. When we define a class, we are interested in a specific set of attributes, as a subset of all possible attributes or actual irrelevant attributes. It should be an accurate representation of a real system, and we need to make the judgment as to what is relevant and what is not (Fig. 2.5). This is difficult because real world phenomena are complex, and the information we have about the system is always incomplete. We can only go by what we know, so our domain knowledge (the understanding of the system(s) we are trying to model), whether it be a software, natural, or human, is critically important.

2.7.2 Object diagrams

Object diagrams (Fig. 2.6) are a logical view of the system at runtime. They are a snapshot at a particular instant in time and can be understood as an instance of a class diagram. Many parameters and variables change value as the program is run, and the object diagram's function is to map these. This runtime binding is one of the key things object diagrams represent. By using links to tie objects together, we can model a particular runtime configuration. Links between objects correspond to associations between the objects class. So, the link is bound by the same constraints as the class that it enforces on its object.

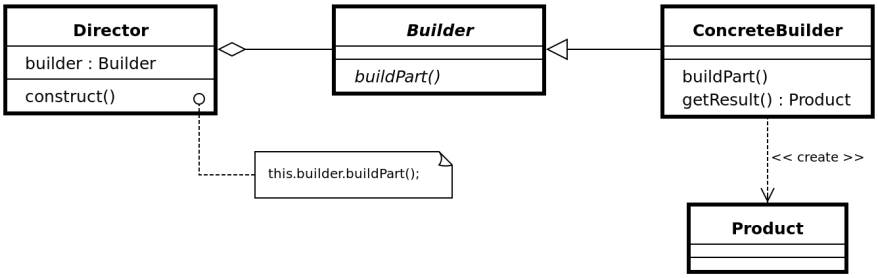

Figure 2.5: The example of a class diagram.

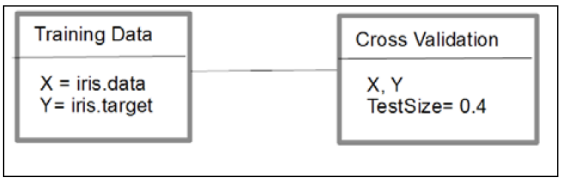

Figure 2.6: The example of an object diagram.

2.7.3 Activity diagrams

The purpose of an activity diagram is to model the system's work flow by chaining together separate actions that together represent a process. They are particularly good at modeling sets of coordinated tasks. Activity diagrams are one of the most used in the UML specification because they are intuitive to understand as their formats are based on traditional flow chart diagrams. The main components of an activity diagram are actions, edges (sometimes called paths) and decisions. Actions are represented by rounded rectangles, edges are represented by arrows, and decisions are represented by a diamond. Activity diagrams usually have a start node and an end node. An example of the activity diagram is on Figure 2.7.

2.7.4 State diagrams

State diagrams are used to model systems that change behavior depending on what state they are in. They are represented by states and transitions. States are represented by rounded rectangles and transitions by arrows. Each transition has a trigger, and this is written along the arrow. Many state diagrams will include an initial pseudo state and a final state. Pseudo states are states that control the flow of traffic. Another example is the choice pseudo state. This indicates that a Boolean condition determines a transition. A state transition system consists of four elements; they are as follows:

- S = s1, s2, . . . : A set of states
- A = a1, a2, ...: A set of actions
- E = e1, e2, ...: A set of events
- y: S(A U E)->2s: A state transition function

The first element, S, is the set of all possible states the world can be in. Actions are the things an agent can do to change the world. Events can happen in the world and are not

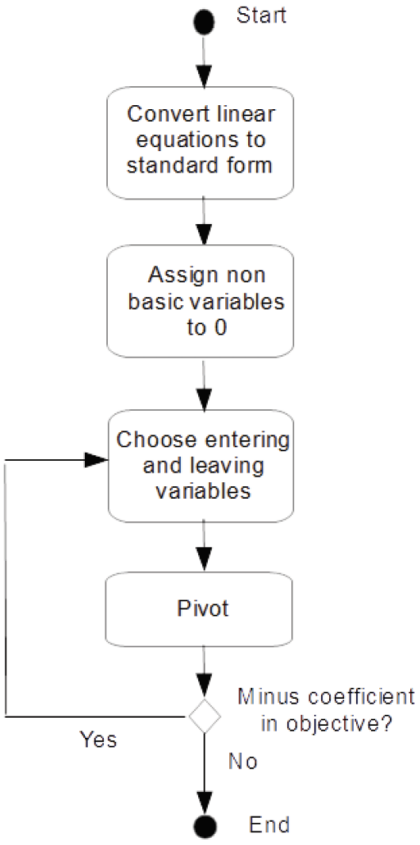

Figure 2.7: The example of an activity diagram.

under the control of an agent. The state transition function, y, takes two things as input: a state of the world and the union of actions and events. This gives us all the possible states as a result of applying a particular action or event.

Consider that we have a warehouse that stocks three items. We consider the warehouse only stocks, at most, one of each item. We can represent the possible states of the warehouse by the matrix (Fig. 2.8).

$$S = \begin{matrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{matrix}$$

Figure 2.8: Matrix representation of the warehouse possible states.

This can define similar binary matrices for E, representing the event sold, and A, which is an action order. In this simple example, our transition function is applied to an instance (s, which is a column in S), which is s' = s + a - e, where s' is the system's final state, s is its initial state, and a and e are an activity and an event respectively. We can represent this with the transition diagram (Fig. 2.9).

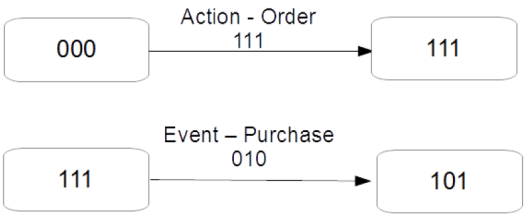

Figure 2.9: Transition diagram example.

2.8 Python for Machine Learning

Python is a versatile general purpose programming language. It is an interpreted language and can run interactively from a console. It does not require a compiler like C++ or Java, so the development time tends to be shorter. It is available for free download and can be installed on many different operating systems including UNIX, Windows, and Macintosh. It is especially popular for scientific and mathematical applications.

Python is relatively easy to learn compared to languages such as C++ and Java, with similar tasks using fewer lines of code. Python is not the only platform for machine learning, but it is certainly one of the most used. One of its major alternatives is R. Like Python, it is open source, and while it is popular for applied machine learning, it lacks the large development community of Python. R is a specialized tool for machine learning and statistical analysis. Python is a general-purpose, widely-used programming language that also has excellent libraries for machine learning applications. Another alternative is Matlab. Unlike R and Python, it is a commercial product. As would be expected, it contains a polished user interface and exhaustive documentation. Like R, however, it lacks the versatility of Python.

Python is such an incredibly useful language that your effort to learn it, compared to the other platforms, will provide far greater pay-offs. It also has excellent libraries for network, web development, and microcontroller programming. These applications can complement or enhance your work in machine learning, all without the pain of clumsy integrations and the learning or remembering of the specifics of different languages.

Python comes equipped with a large library of packages for machine learning tasks. They are not monolithic structures like you would expect from a commercial product, and therefore, understanding the various package taxonomies can be confusing. However, the diversity of approaches of open source software, and the fact that ideas are being contributed continually, give it an important advantage. However, the evolving quality of open source software has its down side, especially for ML applications. For example, there was considerable reluctance on behalf of the Python machine learning user community to move from Python 2 to 3. Because Python 3 broke backwards compatibility; importantly, in terms of its numerical handling, it was not a trivial process to update the relevant packages.

Chapter 3

Dynamic difficulty adjustment

Video games are designed to generate engaging experiences: suspenseful horrors, whimsical amusements, fantastic adventures. But unlike films, books, or televised media in which often have similar experiential goals, video games are interactive. Players create meaning by interacting with the games internal systems [5].

Game developers iteratively refine these systems based on play testing feedback and tweaking behaviors and settings until the game is balanced. While balancing, they often analyze systems intuitively by tracking specific identifiable patterns or types of dynamic activity. It is a difficult and time consuming process [10].

One of the challenges that a computer game developer faces when creating a new game is getting the difficulty "right". Providing a game with an ability to automatically scale the difficulty depending on the current player would make the games more engaging over longer time. In recent years there was an increasing interest in researching how video games can adjust themselves to their players. The aim of this research is to create such an algorithm which might be applied to an educational gamified task assignment. The goal is generally to keep players' attention for as long as it is reasonable. What is a certain way to lose their attention? A straightforward answer is for a game to be boring, be it because of a trivial story, lack of excitement, repetitive too difficult or too easy challenges, etc.

A game might be considered as an interaction between players and their opponent. In this context, assuming their goals are mutually exclusive, difficulty adjustment consists of tuning the skill of the opponent to match the skill of the player. It is possible to estimate the latter and adjust the former based on ranking the moves available to each player [9].

We have an instinct to play because during our evolution as a species playing generally provided a safe way of learning new things that were potentially beneficial for our life.

What constitutes the fun when playing a game? There are three main components in the theories on why gaming is fun: reward, flow and iteration [12]. Reward derives from our intrinsic nature to reward ourselves for doing something. Video games play on this by providing immediate in-game rewards for completing ingame tasks. Flow is the player's ability to become almost a part of the game. Being in the flow means that the player becomes immersed in the game and loses the track of reality and the sense of self. For this to occur in a game the player must be actively involved, concentrated and unaware of realities of time and space boundaries.

Csikszentmihalyi introduced the original concept of flow (Fig. 3.1). He defined it as „*the holistic experience that people feel when they act with total involvement.*" This definition suggests that flow consists of four components: control, attention, curiosity, and intrinsic interest. When in the flow state, people become absorbed in their activity: their awareness

is narrowed to the activity itself; they lose self-consciousness, and they feel in control of their environment. Such a concept has been extensively applied in studies of a broad range of contexts, such as sports, shopping, rock climbing, dancing, gaming and others. Specifically to computer games, a flow is defined as an extremely enjoyable experience, where an individual engages in an on-line game activity with total involvement, enjoyment, control, concentration and intrinsic interest [4].

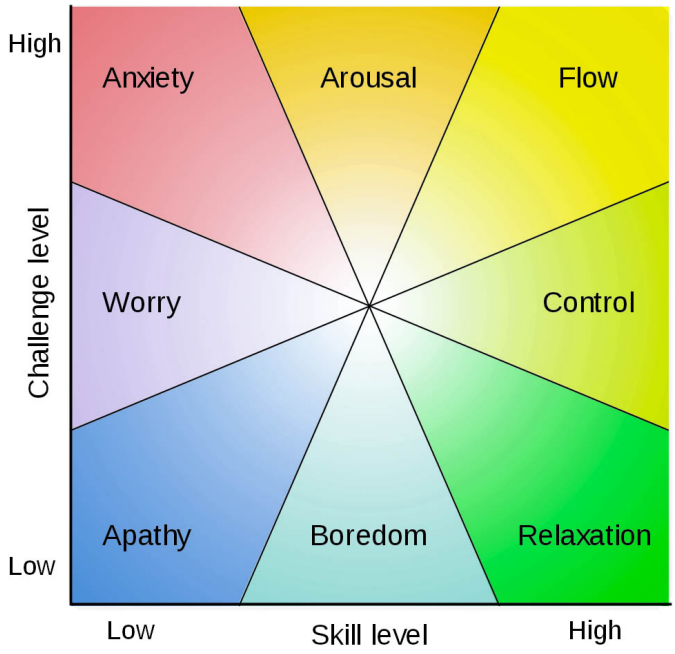

Figure 3.1: Mental state in terms of challenge level and skill level, according to Csikszentmihalyi's flow model.

In other words, the aim is to keep the challenge level always relevant to players' skills. Even when a player enters a game with a low skills level, we should follow their learning curve and adjust the difficulty accordingly. An inherent feature of any challenge (and of the learning required to master it) is its difficulty. Here the difficulty is a subjective factor that stems from the interaction between the player and the challenge: some people find controlling a simulation of a helicopter in a threedimensional space as easy and natural as walking, but most would struggle with it for quite a while before they master it. This example also demonstrates that the perceived difficulty is not a static property: it changes with the time that the player spent learning a skill. In general the more time and effort we invest into learning something new, the better we get at it and the easier subjectively the tasks that exercise this skill get. To complicate things further, not only the perceived difficulty depends on the current state of the player's skills and her learning process, the dependency is actually bidirectional: the ability to learn the skill and the speed of the learning process are also controlled by how difficult the player perceives the task. If the bar is set too high and the task appears too difficult, the player will end up frustrated and will give up on the process in favour of something more rewarding. Then again if the challenge turns out to be too easy (meaning that the player already possesses the skill necessary to deal with it) then there is no learning involved: even though the player accomplishes the task and receives the in-game rewards, she is missing out on that internal reward, the

feeling of joy that the moment of mastery provides. And without it there is no sense of accomplishment, which makes the game appear boring (Fig. 3.2).

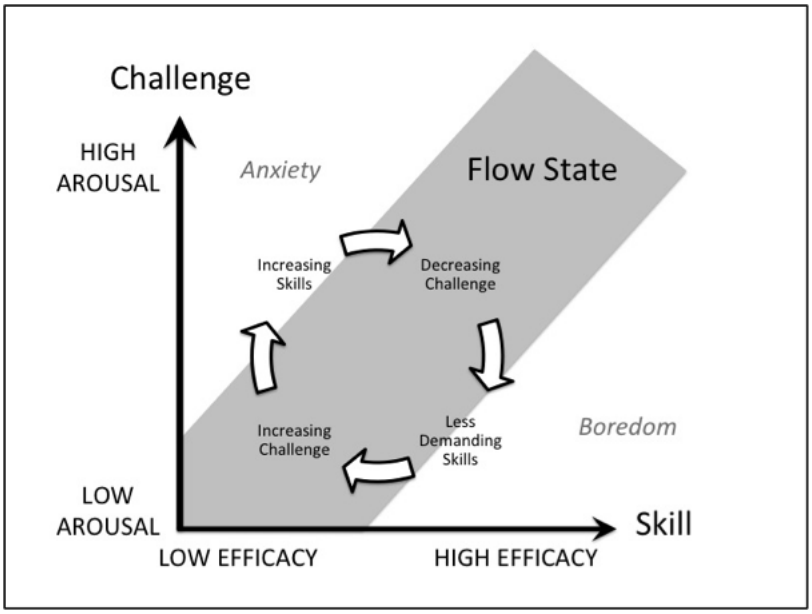

Figure 3.2: Dependency of a flow channel to challenge and skills levels.

And finally, iteration is the games ability to be different upon repetition.

For these reasons the game that strives to be fun should provide the challenges for the player of the "right" difficulty: the one that stimulates the learning without pushing the players too far or not enough. Ideally, the difficulty of any particular instance of the game should be determined by who is playing it at this moment; the game should possess an ability to change the difficulty of its challenges on the fly, in an online fashion.

3.1 Progressive difficulty

The traditional way in which games treat difficulty adjustment is to provide players with a way of controlling the difficulty level themselves. To this end, typical levels would be 'beginner', 'medium', and 'hard'. Such a strategy has many problems. On the one hand, if the number of levels is small, it may be easy to choose the right level but it is unlikely that the difficulty is then set in a very satisfying way.

On the other hand, if the number of levels is large, it is more likely that a satisfying setting is available but finding it becomes more difficult. The necessity of going back and forth between the gameplay and the settings when the tasks become too difficult or too easy disrupts the flow component of the game.

On yet another hand, for game developers, it is not an easy task to map a complex game world into one road map with 3 variables. Constructing such "mapping" requires extensive testing, resulting in time and money costs. Consider also the fact that generally games require several different skills to play them. Providing the computer with an ability to adjust the game to all these skill levels automatically is more user-friendly than offering several settings for a user to set.

3.2 Machine learning algorithms for dynamic difficulty

ML techniques have been widely used in competitive domains, with the focus on finding an optimal strategy which maximizes the payoffs for the agent on most scenarios of competition. It means that the agent must perform as well as possible. Computer games can be seen as competitive environments, however, in this case, it is necessary to achieve a balanced behavior [2].

Game balancing is related to ensuring a good level of challenge in a game, which implies avoiding the extremes of getting the player frustrated because the game is too hard or becoming bored because the game is too easy [7].

Gilleade et al. [3] and Sweetser and Wyeth [11] state that providing players with a personalised, adaptive experience can sustain their attention and keep their interest for longer, which agrees with the argument that games with a personalised dynamic difficulty adjustment (DDA) system are more interesting.

Motivated by this, the aim is to create a mechanism for developing online educational assignments that on the fly provide challenges of the "right" difficulty, i. e., such that players are stimulated but not overburdened. To this purpose, it is investigated how machine learning techniques can be employed to automatically adjust the difficulty of games. A general technique for this problem has natural applications in the huge markets of video games but can also be used to improve the learning rates when applied to serious games or gamification.

According to [2], dynamic balancing of difficulty is divided into three basic rules:

- First, to adapt to the player's profile.
- Second, monitor the performance of the player.
- Third, keep the player interested.

All this without the players realize that the system is making changes to the game. In fact, maintaining the adequate level is a dynamic process, because of the evolution of the player's behavior, as a natural consequence of the experience acquired in playing the game. On the other hand, as user skills can regress (for instance, after a long period without playing the game), regressions of the level are also needed.

There are many different approaches to address DDA. One can control the game environment settings in order to make challenges easier or harder. Although this approach may be effective, its application is constrained to game genres where such particular environment manipulations are possible.

Another approach to dynamic game balancing is to modify the behavior of the Non-Player Characters (NPCs), characters controlled by the computer and usually modeled as intelligent agents. This approach has been innovatively implemented employing genetic algorithms techniques to keep alive agents that best fit the user level. However, it shows some limitations considering skilled users or users with uncommon behavior, as it takes a long time until the agents reaches the user level.

Chapter 4

Conclusion

This essay covers the broad area of such innovative educational methods, as gamification and serious games. Related problems and possible solutions with the help of Machine learning algorithms are described.

Since this is mainly just a theoretical state of the art, the future work lies in the practical are of the implementation of the gamified framework. Some part of it is already done, in a form of a single gamified exercise for learning jQuery in a Web Development course at VUT FIT. Next steps include more testing and evaluation, and also developing more tasks to cover other topics. After this, machine learning should be used to create a variable difficulty for learners.

Bibliography

[1] Acerbi, A.; Lampos, V.; Garnett, P.; et al.: The expression of emotions in 20th century books. *PloS one*. vol. 8, no. 3. 2013: page e59030.

[2] Andrade, G.; Ramalho, G.; Santana, H.; et al.: Automatic computer game balancing: a reinforcement learning approach. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. ACM. 2005. pp. 1111–1112.

[3] Gilleade, K.; Dix, A.; Allanson, J.: Affective videogames and modes of affective gaming: assist me, challenge me, emote me. *DiGRA 2005: Changing Views–Worlds in Play.*. 2005.

[4] Hsu, C.-L.; Lu, H.-P.: Why do people play on-line games? An extended TAM with social influences and flow experience. *Information & management*. vol. 41, no. 7. 2004: pp. 853–868.

[5] Hunicke, R.; Chapman, V.: AI for Dynamic Difficulty Adjustment in Games. 2004. *Association for the Advancement of Artificial Intelligence (AAAI)*: pp. 2–5.

[6] Julian, D.: *Designing machine learning systems with Python*. Packt Publishing Ltd. 2016.

[7] Koster, R.: *Theory of Fun for Game Design: Paraglyph*. 2004.

[8] Kotsiantis, S. B.; Zaharakis, I.; Pintelas, P.: Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*. vol. 160. 2007: pp. 3–24.

[9] Missura, O.; et al.: Dynamic difficulty adjustment. 2015.

[10] Rollings, A.; Adams, E.: *Andrew Rollings and Ernest Adams on game design*. New Riders. 2003.

[11] Sweetser, P.; Wyeth, P.: GameFlow: a model for evaluating player enjoyment in games. *Computers in Entertainment (CIE)*. vol. 3, no. 3. 2005: pp. 3–3.

[12] Tekinbaş, K. S.: *The game design reader: A rules of play anthology*. MIT press. 2006.