Learning User Profiles for Automatic Test of Games

Luis Fernandes
luis.martins.fernandes@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2021

Abstract

Since the increase of the complexity of video games, its getting harder and expensive to fully test, and assure the quality of the game components. To solve this issue, the industry is moving towards the automation of the testing process.

This thesis focuses on the development of an agent that will have the capability of testing different video game features. In addition to this capability, our agent will be able to have different types of behaviour when playing through the game. These behaviors are usually generated through the use of functions or heuristics that are manually created by the game designers. Instead of this method, we proposed a method that will generate the behaviours based on observing a player, this way we are going to create profiles that are closer to the human player behaviour.

Keywords: Player profiles, Inverse Reinforcement Learning, Automated Testing, Max Entropy, Apprenticeship learning

1. Introduction

1.1. Motivation

One of the most important parts of software development, in particular for video game development, is the testing process. In this process the various features and concepts of the game are evaluated, considering a set of defined metrics. This is done with the goal of ensuring the quality of the video game, regarding technical quality and experience.

Typically, this process is performed manually by groups of people, hired by the developer company or, in some cases, by members of the development team. One problem with this manual approach is the process scalability, since the results obtained are related to the number of tests performed and who performs them. Also, to increase the test coverage, it may be necessary to test the game with different players. This scalability greatly affects the time and resources spent during the testing phase.

To solve the escalation problem, the testing process is starting to be automated. Meaning the tests are now being done by software agents, rather than real people. This automation introduced many advantages to the testing process. Regarding the technical quality, with test automation, it allows for a reduction in the feedback cycle of new video game features and accelerates their validation. By increasing the number of tests, the coverage of the testing process will also increases. This will help to test all of video game features, and their overall quality.

1.1.1 Learning Player Profiles

With the removal of real users from the testing process, it will be necessary for agents, to have behaviours that are similar to real users. There are various techniques of creating these said agents. One of those techniques is based on collecting player data. Then use this data in order to make the agent learn the profiles/personas of players, based on their observable behavior. This learning process is defined as apprenticeship learning. Considering that the agent will try to infer the goal of the player and not try to copy it directly. This works as an alternative to having to manually create policy or heuristics for agents to follow.

To carry out this type of learning, the approach followed in this thesis is based on Inverse Reinforcement Learning(IRL).

With these agents, it will be possible to see how different types of players interact on a level. What a player considers most important to play. And finally how do certain levels allow for different types of gameplay. For example, a certain level can cause a player to explore more or be more aggressive.

1.1.2 Profile clustering

As mentioned before, the approach analysed here is based observed data. Although, after we gather the said data, we cannot use it directly on the IRL algorithm. Since different players play differently.

This will cause disturbance in the agent learning, and will result in very poor results.

So first we need to organize the player data. To do this, we are going to use and analyse different clustering approaches. This will create clusters of players, where we will have a different type of behavior in each one.

Then we are going to use the defined player clusters in the IRL algorithm to train the agent.

1.2. Objective

With this thesis, we aim to create agents capable of testing video games and are capable of following different profiles. To replicate these profiles, we are first developing a clustering tool that will be able to group the player data. Then we are going to present an approach based on the Inverse Reinforcement Learning framework. This approach will take the resulting clusters and then replicate the player profile that exists within that cluster.

In the end, we will be analysing the performance of these agents, by comparing them to their respective player data. With this we will see if using this approach is possible to replicate player profiles.

1.3. Thesis Outline

As mentioned before, this thesis is composed of three major parts following the introduction.

On Chapters **??** and **??**, we explain the theory behind this area of research. We briefly present some important concepts such as: Clustering algorithms; We explain in detail IRL, then we delve into a specific IRL algorithm that we are interested in for this project: Max Entropy IRL. Finally, we present some work done in areas that are relevant to the topic of this thesis.

In Chapter **??**, we will analyse the different cluster approaches, present the implementation of the proposed architecture, and present the method used for evaluation.

In Chapters **??** and **??** we will discuss obtained results, from which we will draw conclusions regarding the effectiveness of the approach followed.

2. Background

In this section, we will give a more detailed explanation, regarding what is the method of agent learning, how the Inverse Reinforcement problem can be formulated, bases behind the analysed algorithm, and how such algorithm is capable of generating/capturing a player profile.

2.1. Apprenticeship learning

As Mentioned before, the propose of this thesis is to create an agent, who would be able to learn from human behaviour. This type of learning is usually called Apprenticeship learning. This method via inverse reinforcement learning, will try to infer the goal of the expert behaviour. In other words, it will learn a reward function from observations, which can then be used in reinforcement learning to generate policies. For example, if it discovers that the goal is to hit a nail with a hammer, it will ignore the blinks and scratches from the expert, as they are irrelevant to the goal.

2.2. Markov Decision Process

The base for reinforcement learning and inverse reinforcement learning is the Markov Decision Process formulation.

A (finite-state) Markov Decision Process (MDP) can be represented as the following tuple $M = (S, A, \{P_{sa}\}, \gamma, R)$ where S is S is a finite set of states; A is a set of actions; P_{sa} is a set of state transition probabilities (here, P(s,a) is the state transition distribution upon taking action a in state s); $\gamma \in [0, 1]$is a discount factor; and $R : S \mapsto A$ is the reward function,

2.3. Max Entropy theorem

The principle of maximum entropy states that the most appropriate distribution, to model a given set of data, is the one with highest entropy among all those that satisfy the constrains of our prior knowledge.

2.4. Clustering

Is an unsupervised learning technique where the goal is to divide data points into a number of groups. Such that the data points in the same groups, are more similar to other data points in the same group, than those in other groups.

Therefore, when providing data, that is described by a set of variables, the goal is to identify the constraints that put each record in a specific cluster. In some manner, the goal is just to recognize what are the most important variables, and what are the values assigned to them in each cluster.

We can say that the act of clustering, is similar to classifying. But, in clustering we are discriminating among records, and in classification, we are discriminating as a function of the target variable.

In the book [2] about clustering, the term is general for formal, planned, purposeful, or scientific classification.

This technique is used on a variety of areas for example:

- Economy - where customer segmentation has been its most paradigmatic case.

- Biology - clustering can be applied in phylogenetics for identifying groups of similar organisms, and in transcriptomics, to recognize groups of genes with related expression patterns.

2.4.1 Algorithm types

As we can see, this technique is based on concept of similarity. Which may vary from algorithm to algorithm. In this thesis, we chose to analyze 3 different algorithms, that follow different models.

Connectivity models: As the name suggests, these models are based on the notion that the data points closer in data space exhibit more similarity to each other than the data points that are lying farther away. These models can follow two approaches. In the first approach, they start with classifying all data points into separate clusters and then aggregating them as the distance decreases. In the second approach, all data points are classified as a single cluster and then partitioned as the distance increases. Also, the choice of distance function is subjective. These models are very easy to interpret but lack scalability for handling big datasets. Examples of these models are hierarchical clustering algorithm and its variants.

Centroid models: These are iterative clustering algorithms in which the notion of similarity is obtained through the distance of a data point to the centroid of clusters. K-Means clustering algorithm is a popular algorithm that falls into this category. In these models, the number of clusters required at the end, has to be mentioned beforehand, which makes it important to have prior knowledge of the dataset. These models run interactively to find the optimal solution.

Distribution models: These clustering models are based on the notion of probabilities of all data points in the cluster belonging to the same distribution (For example: Normal, Gaussian). These models often suffer from overfitting. A popular example of these models is Expectation-maximization algorithm which uses multivariate normal distributions.

2.5. K-Means algorithm
As mentioned in the previous section K-means is a centroid based (or a distance-based) algorithm. The goal of this algorithm is to partition n points into k clusters, where each point belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. The first time this algorithm was proposed was at Bell Labs in 1957, in relation to the field of signal processing by Stuart Lloyd as a technique for pulse-code modulation.

This algorithm is divided into 2 steps or phases, the algorithm iterates between this 2 steps until it satisfies the stopping criteria.

- Assignment step - where we assign points to the cluster with the closest centroid
- Update step - where we update the centroids of the new formed clusters

To define the stopping criteria, we can use one of the three stopping criteria to stop the K-means algorithm:

1. Centroids of newly formed clusters do not change
2. Points remain in the same cluster
3. Maximum number of iterations are reached

Unfortunately, this algorithm has some limitations: The fact that the number of clusters, k is an input parameter. An inappropriate choice of k may lead to poor results. One way to solve this, before performing k-means, it is important to run diagnostic checks to determine the number of clusters in the dataset. Another way is to execute the algorithm multiple times with different values of k, then comparing the results.

Other important limitation of k-means is its cluster model. The concept is based on spherical clusters that are separable so that the mean converges towards the cluster center. The clusters are expected to be of similar size, so that the assignment to the nearest cluster center is the correct assignment.

2.6. Expectation–Maximization algorithm
As mentioned in the previous section the Expectation–Maximization algorithm or EM for short, is a model based on distribution.

Before explaining the algorithm, we first need to define what EM is. Em consists on a iterative optimization method, where the goal is to estimate some unknown parameter Θ, given measurement data U. Although, we don't receive some hidden variables J, which we need to integrate. We mainly want to maximize the posterior probability of the parameter Θ given the data U, marginalizing over J as the following equation:

$$\Theta^* = argmax_{\Theta} \sum_{J \in J^n} P(\Theta, J|U)\,. \qquad (1)$$

The intuition behind EM is an old one: alternate between estimating the unknowns Θ and the hidden variables J. This idea has been around for a long time. However, instead of finding the best J ∈ J given an estimate Θ at each iteration, EM computes a distribution over the space J. The algorithm was initially presented in the article [3]. A good reference to EM and its applications can also be found in [8].

Taking into account some of the equations presented in [4] we can simplify the EM algorithm into:

- **E-Step**. Estimate the missing variables in the dataset.

- **M-Step**. Maximize the parameters of the model, in the presence of the data.

As we can see, the technique used in EM is similar to the K-Means technique. Since both are divided into 2 steps, that are related to the assignment step and update step, respectively. Nonetheless, each approach will reach different ends. In the case of k-means, we obtain a hard clustering solution, where a point belongs specifically to one, and only one cluster. In EM we obtain a soft solution, where a point has different probabilities of belonging to different clusters.

In terms of limitations, the EM also has some:

- Similar to k-means, it is necessary to know the number of desired clusters before we start.
- In EN we don't have the guarantee that we will find the globally best solution.
- The algorithm is slow for large datasets.

2.7. Hierarchical algorithm

Hierarchical clustering is another cluster technique that we decided to explore. The most dominant approach for this algorithm, has been the Agglomerative strategy. This strategy consists of a "bottom-up" approach, where each observation will start in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.

Before we start this algorithm, there are things that we need to consider first, beyond the number of clusters we want.

Firstly, since we want to group the data, we need a way to measure each element size, and their distances relative to each other in order to decide which elements belong to a group. This choice of distance metrics, should be made based on the theoretical concerns of the domain under analysis. In other words, the distance metric needs to define similarity in a way that is sensible for the field of study. The most used metrics are Euclidean, Manhattan and Chebyshev distances. Where the Euclidean distance is generally the most used one. In the example shown, the metric we are going to use, is Euclidean.

Secondly, we need to determine from where we are going to compute the distance. This is defined as the linkage criteria, which has different approaches. For example, in the single-linkage, the distance is computed between the two most similar parts of a cluster. In the complete-linkage, are taken in consideration the most different parts of a cluster. Or we can use the average-linkage that uses the center of the clusters.

Similarly to the distance metrics, when choosing a linkage criteria, we should analyze the domain of application.

Seeing that, this algorithm does not use random centroids for its computation. Saying that, we can affirm that hierarchical clustering consists of a deterministic process, meaning cluster solution won't change when the algorithm is executed multiple times, with the same input data.

2.8. IRL

The main objective in is to obtain a decision process to produce a behavior in a way to maximize some predefined reward function.

In the inverse reinforcement learning initial defined by [7], tries to flip the goal of a traditional reinforcement learning (RL) scenario, instead of learning a behaviour. The objective is to learn the goals, values or rewards of an agent, by observing his behavior.

The main purpose of this approach was to change the perspective of the traditional RL tasks. In RL, the agent aims to maximize a known reward function. With IRL, the main objective of the agent is to find to the reward function based on observed behaviour.

An IRL task can be defined ruffly as the following:

Given:

- 1) measurements of an agent's behaviour over time, in a variety of circumstances
- 2) if needed measurements of the sensory inputs to the agent
- 3) if available, a model of the environment

Obtain: The reward function being optimized

Later in [5] an alternate approach based on Inverse Reinforcement Learning was proposed. Here the proposed strategy consists on matching feature expectations (Equation 2) between an observed policy and a learner's behavior.

$$\sum_{Path\zeta_i} P(\zeta_i) f_{\zeta_i} = \check{f} \tag{2}$$

Then was demonstrated that with this matching it was possible for an agent to achieve the same performance, as if the agent were in fact solving an MDP with a reward function linear in those features.

The advantage of using this framework, is that the reward function has more value to an agent, since the reward function can be migrated from task to task. When applying the framework to video games, we can change the first item with video game runs performed by players, and give representation of the game environment. By doing this, the agent will get the reward function behind the players actions, and their limitations that will be represented in the different runs that the algorithm will receive. This will result in a better understating of a player profile. Although, in this approach, it will still be necessary manual testing of the game, to retrieve the user maneuvers. However, in the end, the overall number of manual tests will be lesser than, the situation where the tests are all done manually. Considering that, once the agent obtains the rewards of the player, it can automatically test the game as a player.

3. Implementation

In this section it is explained the approach taken to resolve the defined problem. This section is dived into 5 subsections. First, we will provide specifications for the software used in this thesis. Then in the other subsections, the other parts of the solution are going to be analysed. Namely, the testing environment, the different clustering algorithms, the agent proprieties, in addition to their Profiles. Ending with an explanation on how the validation process for the project, will be carried out.

3.1. System Requirements

The most important software used in this thesis is the programming language Python, since, all of important computations, are simplified using Python modules. We give special attention to the Numpy module that facilitates matrix operations and computations. The clustering algorithms used in this thesis also come from a python module, in this case sklearn. We used this module due to the variety of different types of clustering algorithms provided by the module, and also, it has methods to perform clustering analysis. The game environment was also built in Python, through the use of the Pygame engine. In addition, to present some of the results in a more visual way, Matplotlib module was used.

3.2. Game Environment

As mentioned in the acknowledgements section, this thesis is based on a game developed by Pedro Fernandes. In Figure1 we can see a screenshot of the game environment.

The game is named infinite game. It consists on a 2d top down inspired by the classic Legend of Zelda games [6]. In this game the player can explore a level with the main objective of reaching the flower at the end. When navigating through the level, the player will encounter enemies. Then the player can fight these enemies or escape from them. Finally, in the game, the player can collect optional coins.

The game is divided into 3 levels, which have same layout as we show in figure 2. In addition to the layout of the levels, the player's starting position, and the objective's position, are the same.

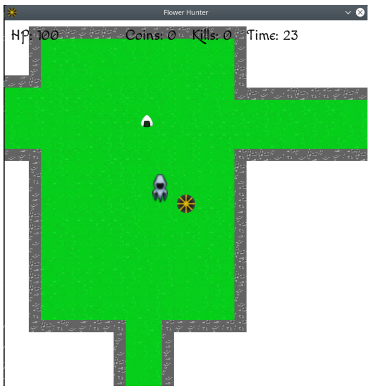


Figure 1: Infinite Game.

The main differences between the levels are related to the number of objects that the player can interact with. In this case, the number of enemies, the number of coins, and finally the health pickups. Furthermore, this also has an affect on the difficulty of each level. Making the level 1 the easiest one, and the level 3 the hardest.

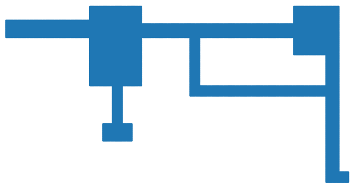


Figure 2: Infinite Game level layout.

There were several reasons for choosing this game as an analysis environment. The first reason is related with the simplicity of the game. Since, in terms of mechanics the game isn't very complex, and it's world has only 2 dimensions. This makes it easier for us to create a symbolic representation of it, mainly available actions, and possible agent states.

Secondly, there is already some available data. Since this game is part of the iv4XR project, and is being used for other project components, we already have access to player data, that was collected in previous experiments. The said data is composed of the following:

- For each level, we have 90 player traces, making a total of 270 traces.
- Each player trace is dived into 3 parts:

  position all positions the player has gone through.

  actions all actions the player performed.

preceptor traces of relevant game data, for example, distance to the final objective.

In order to train the agents, we are going to use all available traces. However, as we mentioned before, the IRL algorithm that we are going to use, is prone to noise. With this in mind, instead of using the data directly, we are going to perform clustering, with the goal of producing smaller datasets.

3.3. Clustering

As mentioned, all algorithms used here, came from the sklearn module. When performing the clustering as we mentioned in section 2.4 one common problem on the 3 algorithms, is that we first need to define a number of clusters k. Basically, k needs to be an input value, and it is hard to define a value for k. Therefore, to overcome this problem, we decided to run the 3 algorithms several times, with different k values. Then, compare the different results according to specific metrics. So, first we define an array with the various values for k. $array = [2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]$

Now we define the relevant features that are going to be used for clustering. For this we decided to use 4 that we defined as the most relevant when establishing a player profile. Those are:

- Distance to objective
- Remaining life
- Number of coins collected
- Number of enemies defeated

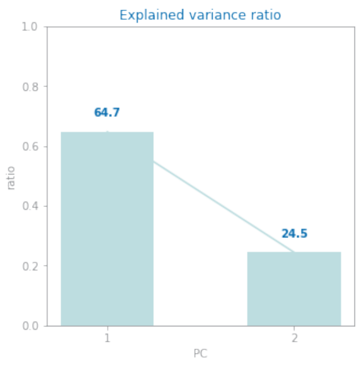
Since the data has 4 dimensions, we decided to test if it was possible to reduce the dimension space, mainly to simplify the readability of the different clustering solutions. To do this we decided to test Principal Component Analysis (PCA), to see if with only 2 variables it is possible to represent most of the data.

As shown in figure 3 with 2 features, PCA covers 89% of the not normalized data, and 91% of the normalized data.

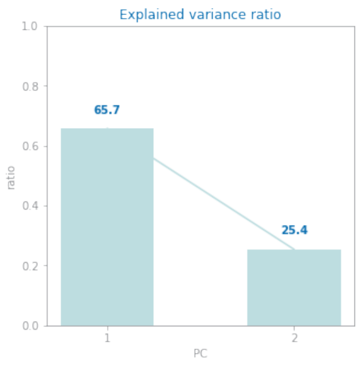
Now we select the metrics that we are going to use with purpose of comparing the different algorithms. We decided on two metrics, the Mean Square Error (MSE) and the Silhouette Score. Both metrics are already implemented in the sklearn module.

3.4. Agent Structure

For the creation of agents, we decided to follow an approach similar to [9]. We decided to divide the agent's behavior into two main components, so that the agent has two types of behaviours when playing trough the level. These two modes are: the **navigation mode** and the **combat mode**. This division was created for 2 main reasons:



(a) No normalization



(b) Normalized

Figure 3: PCA variance.

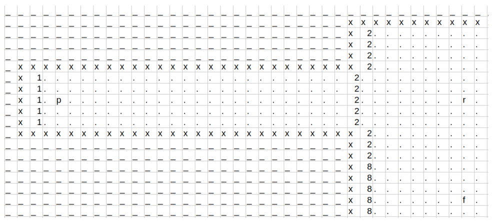
- Since the levels have enemies whose position is not fixed, it would make somewhat complex, represent agent states if we consider the various enemy positions.
- This division simplifies the combination of different behaviors. For example it will be possible to have an aggressive agent, who wants to explore the level or go straight to the end.

The transition between these two modes is mainly related to the distance between the agent and the closest enemy. In essence, the agent will switch to the combat mode when it gets close to an enemy.

3.4.1 Navigation mode

This mode is related to how the agent navigates through the different levels. Here, the objective was to represent a level using a GridWorld approach. In this approach, the state space of the MDP will match all valid positions of a level.

To define MDP state space, we decide to take the level structure, defined as a csv file. In figure 4 we show a part of the csv map file. Then we divided the map into different areas as later shown in figure **??** (a). This division was done to reduce the size of the state space.



```
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ x x x x x x x x x x x
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ x 2 . . . . . . . . .
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ x 2 . . . . . . . . .
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ x 2 . . . . . . . . .
_ x x x x x x x x x x x x x x x x x x x x x x x x x x x 2 . . . . . . . . .
_ x 1 . . . . . . . . . . . . . . . . . . . . . . . . 2 . . . . . . . . . .
_ x 1 . . . . . . . . . . . . . . . . . . . . . . . . 2 . . . . . . . . . .
_ x 1 . p . . . . . . . . . . . . . . . . . . . . . . 2 . . . . . . . . r .
_ x 1 . . . . . . . . . . . . . . . . . . . . . . . . 2 . . . . . . . . . .
_ x 1 . . . . . . . . . . . . . . . . . . . . . . . . 2 . . . . . . . . . .
_ x x x x x x x x x x x x x x x x x x x x x x x x x x x 2 . . . . . . . . .
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ x 2 . . . . . . . . .
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ x 2 . . . . . . . . .
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ x 8 . . . . . . . . .
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ x 8 . . . . . . . . .
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ x 8 . . . . . . . . .
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ x 8 . . . . . . . f .
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ x 8 . . . . . . . . .
```

Figure 4: CSV map file.

To extract the states from the file, the method used was as follows. First, we store the positions of all important cells of the file. These are represented as ".", "P", numbers, among others. These cells, when they are registered, are stored in lists according to the area where they belong. Secondly, we obtain the cell position that corresponds to the player's starting position, identified in the file by P. Then for each area we will normalize the values of the positions that belong to its list, according to the value of P. For example if P is (10, 10) and A is (14, 10), then after normalization P is (0,0) and A is (4,0). Finally, we create a Python dictionary where each key corresponds to an area id and the value points to a list of all positions of the said area.

It is important to mention that the positions that are extracted from the csv do not correspond to a specific position on the game map. Instead they encompass several positions on the map.

The final state space has a total value of 1285 states with the following area division:

Table 1: States per area.

Area	NºStates
area 1	125 states
area 2	197 states
area 3	126 states
area 4	202 states
area 5	255 states
area 6	105 states
area 7	147 states
area 8	128 states

For the action space, we consider 5 simple actions:

- The four directions for movement UP, DOWN, LEFT and RIGHT.
- An "empty" action STAY since when analyzing the player's data, we saw that the player is sometimes in the same position.

As for the transition matrix between states, we define that all of the actions as deterministic. Meaning an action performed in a specific state can only transit to only one state. Because in a video game scenario, actions with an associated error don't usually exist. As an error we can consider selecting the action RIGHT and instead of going right we go left.

3.4.2 Combat mode

For the structure of combat MDP, we decided on a more abstract solution. Since during combat, the most important features, are the location of the enemy and the current player's life. We define the state space as a combination of these 2 elements, giving a total of 12 states as shown in table 2.

Table 2: Combat mode states.

dist to enemy \ Life %	0-29	30-59	60-89	90-100
≤ 40	1	2	3	4
≥ 40 and ≤ 80	5	6	7	8
≤ 100	9	10	11	12

For the available actions we decided on 5 possible actions:

- moving towards the enemy
- moving away from the enemy
- move towards and attack the enemy
- attack the enemy
- stay

With these actions, it is possible to replicate several behaviors that are related to combat. Such as, we can have one that only escapes from enemy, or one that goes straight to the attack.

3.5. IRL

For the implementation of the IRL algorithm, we followed the Max Entropy as it was presented in [10]. Inspired also by [1] that had already an implementation of this same algorithm defined here as MaxEnt. To his code, we made some modifications to deal with the concept of level areas. To execute the MaxEnt IRL the most important function to consider is the following:

def irl(feature_matrix, n_actions, discount, transition_probability, trajectories, epochs, learning_rate, area_mdp_id, profile_id):

Where the attributes correspond to the following

- **feature_matrix**: matrix for state features
- **n_actions**: number of available actions
- **discount**: discount factor of the MDP
- **transition_probability**: a (Action, State, State) matrix

- **trajectories**: a set of trajectories based on observed behaviour

- **epochs**:the number of epochs used to train

- **learning_rate**:the learning rate of algorithm between 0 and 1

- **area_mdp_id**: id of the current area under analysis

- **profile_id**: id of the current profile under analysis

For this project we decided on using an Identity matrix for the feature matrix, for the training epochs we decided on a value of 60. In relation to trajectories, we define a trajectory as being a list of following: (state, action, reward), here the state is the cell position, the action is one of the five defined in section 3.4.2, the reward is the value extracted from a basic reward function where most of the states have a value equal to 0, the cells where are coins have a value of 0.5 and the final objective has a value of 1.

3.6. Verification and Validation

To test the approach analysed in this thesis we defined a process that is divided into 3 parts:

1. IRL results

2. Profiles vs Traces

3. New level

In the first part, we are going to compare the learned policies to the player behaviour. This process will be based on 2 metrics: similarity and probability. In the case of similarity, for each state, we are going to analyse the average of player actions. Then compare it to the action in the agent policy. For the probability, the process is similar to the similarity, but, instead of counting the actions that are equal, we will calculate the probability of the agent performing the action sequence described by the average player behaviour. To validate these results, we consider similarity superior to 50% to be positive. For the probability we expect very low values, having said that we consider an exponent -35 to be positive.

On the second part, we are going to compare the performance of the agent and compare it to the player behaviour. To do this, we are going to make the agent play the average level of the traces used to train it. For example, if profile 0 has the most traces of level 1 playthroughs, we should test it on level 1. Then we are going to measure its performance based on the following features: remaining life or HP, the number of coins collected, number of kills or enemies defeated, the time taken to complete the level, and the percentage of map exploration. In case the agent gets stuck in a part of the map, we will set a limit on the time taken.

Finally, we are going to take the different profiles and test them on a new made level. This new level will have a similar layout to the previous ones. Here, we only change the number and position of enemies and coins. Then we are going to evaluate the agents using the previous features.

4. Results

In this section we will analyze the results obtained for the different approaches taken. But first we will describe the baseline problem

As mentioned in the previous sections, the objective of this work is to create agents that are capable of having different profiles, making it possible for the agents to be able to play the game in different ways. It is also important to test the profile learned at the level used in the training, in order to compare it's performance with that of real players. Also it is important to test it in a level different from the one used in training.

First, we need to consider the level layout, as mentioned before, the level was divided into 8 areas. In figure 5 we present the final division of the level into areas and the labeling on them.

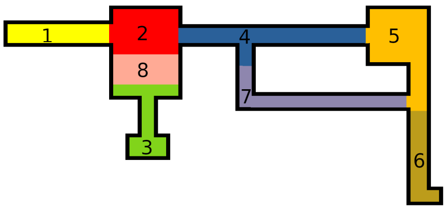


Figure 5: Level map areas

4.1. Solution 1

For solution 1, we decided to use the not normalized data.

From the table 3, as expected, we can see that only profile 4 didn't reach the level objective since its remaining HP is equal to 0. We can also see that most of the profiles captured here did not explore much of the level, most of them move directly to the objective since their map coverage is inferior to 40%. Although the performance of the agents was very similar to that of the player behaviour. We can also see that the agents tend to be more aggressive that the player, since in all cases the agents defeated more enemies than the players average.

From the table 4, when playing the new level, we see that the results are not as good. We can see that only 1 of the agents was able to finish the level, 1 of them died, and the other 3 became stuck. It is interesting that 1 of the 3 that got stuck is the profile that does not have knowledge of the area where the objective is, as was expected. One other that also got stuck, was the one that got lost in the previous experiment. In terms of percentage of the map explored, the values obtained here are very similar to the previous levels.

Table 3: Solution 1 Base Performance Results Table.

agents level	time	coins collect	enemies killed	Hp	map coverage
P0 - l2	45	0	5	45	31.15%
T0 - l2	42	5	1	20	20.99%
P1 - l1	28	1	0	100	22.04%
T1 - l1	30	1	0	100	21.18%
P2 - l3	11	0	4	0	16.67%
T2 - l3	24	0	1	0	15.34%
P3 - l2	78	0	3	25	32.39%
T3 - l2	53	8	2	90	25.61%
P4 - l2	200	15	6	30	31.3%
T4 - l2	97	18	5	30	32.46%

Table 4: Solution 1 Extra Performance Results Table.

agent	time	coins collect	enemies killed	Hp	map coverage
P0	200	4	5	40	31.30%
P1	26	8	4	70	24.45%
P2	200	8	5	70	16.98%
P3	25	3	3	0	23.29%
P4	200	12	5	100	35.5%

4.2. Solution 2

For solution 2, we decided to use the normalized data. Since, we see that with the not normalized data it is harder to compare the behaviours between levels, even though all levels have the same layout, they have different elements.

From Table 5, we see that the performance of the agents shows similar results to that of the player average. As expected we see that 3 profiles didn't reach the level objective, since its remaining HP is equal to 0. An interesting observation taken from this results is that even though the agents map coverage is inferior to 40%, meaning that they did not explore much of the level, all of the agent values are slightly higher than players. Although, most values related to the collection of coins are smaller. Similar to the previous solution, we see that the agents tend to be more aggressive that the player, since in all cases the agents defeated more enemies than players average.

From Table 6, when playing the new level, the results here are not as bad as the previous solution. We see here that 3 agents got stuck, 1 of them got stuck in the previous experiment, the other 2 are agents that don't possess knowledge of the area where the objective is, meaning they are incapable to finish the game. In addition to this, we also have different agents losing. But, these were trained based on the level 1 witch only has 1 enemy, which can have an impact on the agent combat performance.

Table 5: Solution 2 Base Performance Results Table.

agents level	time	coins collect	enemies killed	Hp	map coverage
P0 - l2	200	10	1	90	22.04%
T0 -l2	38	10	0	0	21.17%
P1 - l1	28	1	0	100	21.57%
T1 - l1	34	1	0	100	22.72%
P2 - l3	13	0	5	0	17.92%
T2 - l3	22	0	2	0	15.96%
P3 - l2	45	0	5	65	31.30%
T3 - l2	43	8	1	85	24.87%
P4 - l1	27	0	0	100	23.83%
T4 - l1	64	2	1	80	28.53%
P5 - l3	26	0	8	0	22.51%
T5 - l3	26	0	2	0	17.62%
P6 - l2	89	0	7	0	34.34%
T6 - l2	59	6	6	0	23.26%
P7 - l1	64	0	0	100	35.11%
T7 - l1	56	1	1	95	25.76%
P8- l1	28	0	0	100	23.75%
T8 - l1	37	0	0	100	22.27%
P9 - l2	46	0	5	35	31.07%
T9 - l2	80	11	4	85	28.71%
P10 - l2	62	0	4	30	32.23%
T10 - l2	85	10	4	10	33.56%

5. Achievements

After analysing the obtained results, we can draw some conclusions regarding what was achieved on the work done during this thesis development. First, we demonstrate that it is possible for an agent, to learn only based on observed player behaviour. And this was enough to make agents play in a slightly different way among them. It was also possible to make the agents play on a different level, other that the ones that were used to train

Table 6: Solution 2 Extra Performance Results Table.

agent	time	coins collect	enemies killed	Hp	map coverage
P0	200	10	5	10	28.42%
P1	27	12	4	100	24.76%
P2	121	11	5	0	26.55%
P3	67	16	5	65	32.47%
P4	38	10	4	0	27.18%
P5	200	8	4	100	28.97%
P6	200	18	6	70	33.25%
P7	38	0	3	0	23.75%
P8	28	17	5	70	24.76%
P9	41	16	5	55	31.15%
P10	78	18	6	5	36.36%

it. Although, the disadvantage is that this new level needs to have the same layout has the previous ones.

Even though the model we used here was the simplest one, and has some limitations, making it not the best one. For example, it tends to make agents less interested in level exploration, and punish those who try to explore. Even thought with those constrains, the obtained results were positive, where most of the agents performed similar to the respective player averages.

The division of the agent behaviour into navigation and combat proved to be a good addition. This division helped to simplify the structure of agents, and had an impact on the learning process. Thanks to this when training the agents, we could divide the observed behaviour into parts, which allow the training to be focused on a specific part. Other good addition was the area division, and this helped by reducing the time taken by the agents to learn a profile. Although, it creates a new problem, when diving the map we might lose the level structure. In addition, to define these areas, it is necessary to be careful with the number of areas that are defined and the size of them. And this design might not be suitable to every video game.

When talking about the IRL algorithm, choosing the use of the MaxEnt algorithm turned out to be a good choice. This algorithm was already prepared to receive the player behaviour in the form of a list of trajectories. Although, this algorithm has some limitations such as: It was necessary for the all the trajectories, to be of the same size, so it was necessary to add extra elements to some of the trajectories, creating duplicated ones. Another problem with this algorithm is the time it takes to provide results, during the research, we saw that the default value for the training epochs was 60, and this value with the original level structure simply takes a long time to obtain results. To solve this we created the separation of a level into areas.

Acknowledgements
I would like to thank everyone who was essential in this academic phase.

A big thanks to everyone.

References

[1] M. Alger. Inverse reinforcement learning, 2016.

[2] J. A. Hartigan. *Clustering algorithms.* John Wiley & Sons, Inc., 1975.

[3] H. O. Hartley. Maximum likelihood estimation from incomplete data. *Biometrics*, 14(2):174–194, 1958.

[4] T. Minka. Expectation-maximization as lower bound maximization. *Tutorial published on the web at http://www-white. media. mit. edu/tpminka/papers/em. html*, 7:2, 1998.

[5] A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, page 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[6] Nintendo. Legend of zelda, 1986.

[7] S. Russell. Learning agents for uncertain environments (extended abstract). In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 101–103. ACM Press, 1998.

[8] M. A. Tanner. The data augmentation algorithm. In *Tools for Statistical Inference*, pages 90–136. Springer, 1996.

[9] B. Tastan and G. Sukthankar. Learning policies for first person shooter games using inverse reinforcement learning. In *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'11, page 85–90. AAAI Press, 2011.

[10] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, page 1433–1438. AAAI Press, 2008.