Intelligent Verification/Validation for XR Based Systems

Research and Innovation Action

Grant agreement no.: 856716

D4.2 – 2nd prototype of SETAs

iv4XR – WP4 – D4.2

Version 1.6

December 2021

Project Reference	EU H2020-ICT-2018-3 - 856716
Due Date	31/12/2020
Actual Date	29/12/2020
Document Author/s	Manuel Lopes, Marta Couto, Pedro Fernandes, Carlos Martinho, Rui Prada, Luís Fernandes (INESC-ID), Saba Ansari (UU)
Version	1.6
Dissemination level	Public
Status	Final

This project has received funding from the European Union's Horizon 2020 Research and innovation programme under grant agreement No 856716

Document Version Control			
Version	**Date**	**Change Made (and if appropriate reason for change)**	**Initials of Commentator(s) or Author(s)**
1.0	03/12/2020	Initial document structure and contents	MC, PF
1.2	10/12/2021	Details on Appraisal model of emotions for Automated PX testing	SA
1.3	15/12/2021	Details on behavioural generation, difficulty estimation and interactive narratives testing.	PF
1.4	21/12/2021	Details on creating personas	LF
1.5	22/12/2021	Changed the order of the sections	PF
1.6	27/12/2021	Minor corrections	RP

Document Quality Control			
Version QA	**Date**	**Comments (and if appropriate reason for change)**	**Initials of QA Person**
1.3	18/12/2020	Document review	IS
1.6	27/12/2020	Document review	RP

Document Authors and Quality Assurance Checks		
Author Initials	**Name of Author**	**Institution**
ML	Manuel Lopes	INESC-ID
MC	Marta Couto	INESC-ID
PF	Pedro Fernandes	INESC-ID

CM	Carlos Martinho	INESC-ID
RP	Rui Prada	INESC-ID
LF	Luís Fernandes	INESC-ID
SA	Saba Ansari	UU
IS	Ian Saunter	GW

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This deliverable is a software deliverable. In this document we provide an overview of the state of the work on WP4, software links, and descriptions. We annexed papers and reports that describe in more detail the development and results of the software.

Acronyms and Abbreviations

SETA	Socio-Emotional Test Agent
SUT	System Under Test
XR	Extended Reality
UX	User Experience
RL	Reinforcement Learning

INTRODUCTION

Work Package 4 focuses on developing socio-emotional test agents (SETAs) to aid the systematic assessment of User Experience (UX) of Extended Reality (XR) systems. The SETA's will use well-established emotion appraisal theories and models to assess the emotional state; they will cover social and behavioural variability to simulate a wide range of users; and use a progression model allowing it to appraise UX-relevant user states over time.

To achieve these goals, we need to build several components. To create these components, we need to understand the social and emotional experience of the user. We need comprehensive profiles with social properties of different types of users. We also need information on how the scenarios or type of interaction impact the user. We can then use the profiles to test software automatically with the developed components.

Work on this task is on track. Task 4.2 was slightly delayed due to pandemic constraints that made the planned user studies impossible to conduct. We were planning on collecting physiological measures which required close contact with participants.

In this period the work was divided into four main components:

- study and development of models of user experience and social components of interaction
- study and development of behavioural models that fit the task of testing UX
- study and development of models of difficulty estimation of scenarios
- study and development methods for modelling user profiles

SOFTWARE INCLUDED IN THIS DELIVERABLE

APPRAISAL MODELS FOR UX ASSESSMENT

Machine Learning and the PAD Model of Emotion

The previous implementation of the UX testing agent (D4.1) relied on a two dimensional affective model, based on the dimensions of Arousal and Valence. These two dimensions were updated following a rule-based approach in order to represent the expected affective state of a user. To improve upon this approach, we decided to use a 3-dimensional emotional model, the PAD model of emotion [1], which describes human emotions based on three dimensions: Pleasure; Arousal; and Dominance. We further decided to use machine learning (ML) instead of rules to predict the expected emotional state of the agent.

By using machine learning instead of rules to model the evolution of the emotional dimensions, the predictions will be based on that which real users reported feeling in similar or comparable situations. With this, we aim to achieve a better predictive accuracy and a more reliable model. A representation of the machine learning process used can be found on Fig. 1.

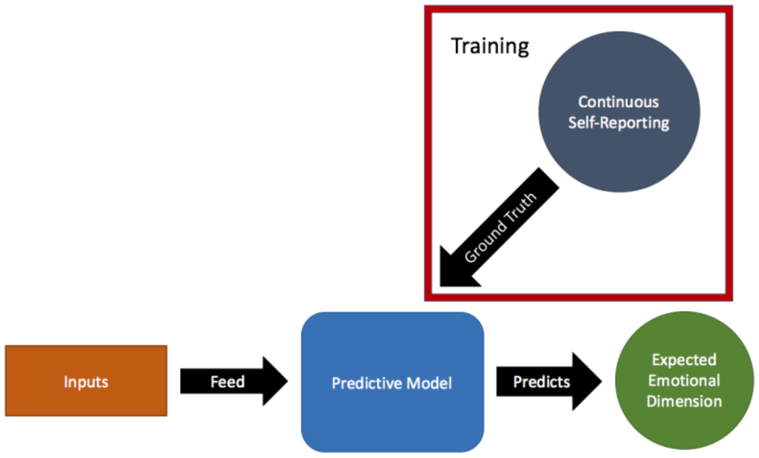


Figure 1: A graphical representation of the machine learning approach.

The use of the PAD model of emotion improves the granularity of the prediction, adding an extra dimension, Dominance, to the two dimensions that were previously used (the previously used Valence dimension being comparable to the Pleasure dimension). This means that instead of using a core affect model, we are using a full emotional model, which has been used to classify a considerable part of the human emotional spectrum [1].

To implement the previously mentioned improvements, a different game was used. The game first used was too simple and failed to provide a good coverage of the emotional space. There were no dangerous elements in the game, for example, which meant that it would be unlikely that a user would experience fear.

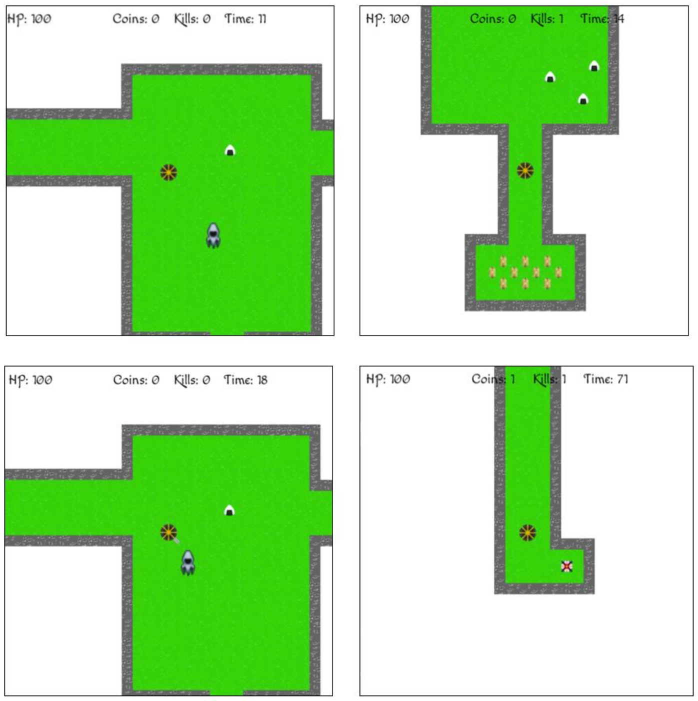

HP: 100 Coins: 0 Kills: 0 Time: 11
HP: 100 Coins: 0 Kills: 1 Time: 14
HP: 100 Coins: 0 Kills: 0 Time: 18
HP: 100 Coins: 1 Kills: 1 Time: 71

Figure 2: Screenshots of the "Flower Hunter" game. The player, represented by a black and yellow ball, needs to traverse a field riddled with enemies, health providing rice cakes, and coins, in order to find a pink flower (shown on the bottom right screenshot). Once the player finds this flower, the game is over and the player wins. If the player loses all its health points by touching enemies, then the game ends and the player loses. The player can use a sword to fight and kill the enemies (shown in the bottom left screenshot).

The game used, named "Flower Hunter", was inspired by old-school top-down 2D games like Legend of Zelda. It is easily modifiable, fast-running, compatible with Python machine learning libraries, and ultimately entertaining enough to motivate users to play it. Screenshots of the game can be found on Fig. 2. This game is still far from a complex XR system, but the studies being done here have never been done on simpler games or applications, so we have been forced to build the foundations of the research ourselves. Once we have proven that our approach works for a simpler game, we can expand and apply it to more complex test cases, as are the iv4XR case studies (Space Engineers, etc…).

With a test-bed game and emotional model decided upon, it was time to define the inputs and outputs of our machine learning model. The inputs would have to encapsulate all relevant information about the game, whereas the outputs needed to relate to the three dimensions of the PAD model of emotions.

Pertaining the inputs, we decided to use numerical values related to the player and the different objects present in the game. The list of the inputs used is as follows:

- Distance to closest enemy
- Distance to closest rice cake (a collectible)
- Distance to closest coin (a collectible)
- Number of enemies in view
- Number of rice cakes in view
- Number of coins in view
- Sum of value/distance of enemies
- Sum of value/distance of rice cakes
- Sum of value/distance of coins
- Seconds since seeing enemy
- Seconds since seeing rice cake
- Seconds since seeing coin
- Distance to objective
- Health Points
- Coins gathered
- Kills
- Damage done

To train a machine learning model, we required information about the three emotional dimensions of a player as he traversed the game, as we believed it would be more meaningful to the testers and designers to know how the emotional dimensions evolved over time and space opposed to only having a final estimate of the value for each dimension for the totality of the interaction between the user and the system. Furthermore, the closest to continuous this information could be, the better, as it would allow us to create a finely grained model.

To achieve this close to continuous annotation, user questionnaires couldn't be used. Physiological data was an option, and might still be used in future work, but it was intrusive, required specialised equipment, and we found no studies directly correlating physiological data with the 3 dimensions of the PAD model. As such, we decided to use continuous, after the fact, annotation, inspired by such works as [2] and [3].

This annotation worked as follows. The user, after playing a given level of the "Flower Hunter" game, was asked to annotate one of the PAD emotional dimensions. She did so by seeing a recording of the level he had just played while using the up and down arrows on a computer keyboard to control a line on the screen, which represented the evolution of the emotional dimension throughout the traversal of the level. Two screenshots taken during the annotation process can be found on Fig. 4.

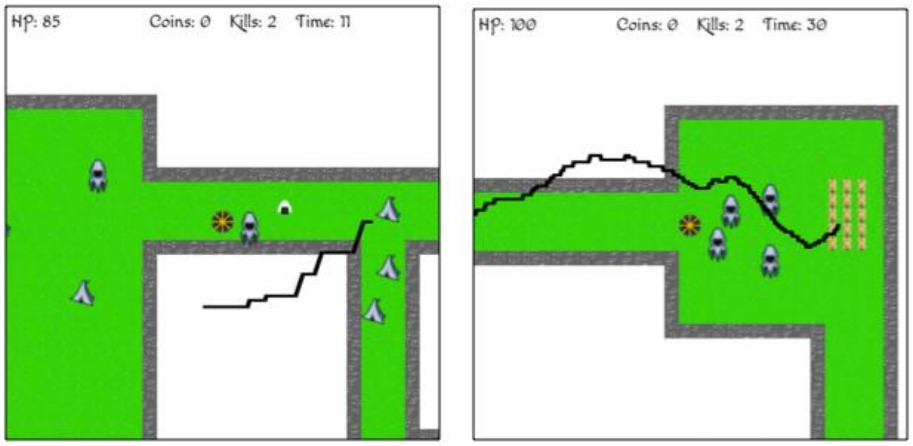


Figure 4: Screenshots of the continuous annotation process used. The black line that is seen on the screen was controlled by the user, going up or down according to the perception the user had of the dimension being annotated having increased or decreased, respectively.

We chose to have each user annotate only one of the PAD dimensions. We did this both to spare the user 3 consecutive annotations after playing a single level, but also to ensure the user kept in mind the dimension that she was annotating without getting confused and inadvertently mixing the dimensions. By annotating a single dimension, the user only had to remember a single definition for the dimension being annotated, thus, in principle, ensuring more reliable annotations. This came, however, at the cost of having only one of the dimensions annotated for each user trace we had.

We conducted a study to collect data with 88 participants playing three "Flower Hunter" levels and self-reporting their levels of a given PAD dimension. Each participant was randomly assigned a dimension at the beginning of the experiment and given a definition of said emotional dimension. The participants could only proceed with the experiment once they confirmed that they understood the definition for their assigned emotional dimension. The levels used for the experiment can be found on Fig. 3. In the end, we had 264 annotated gameplay traces.

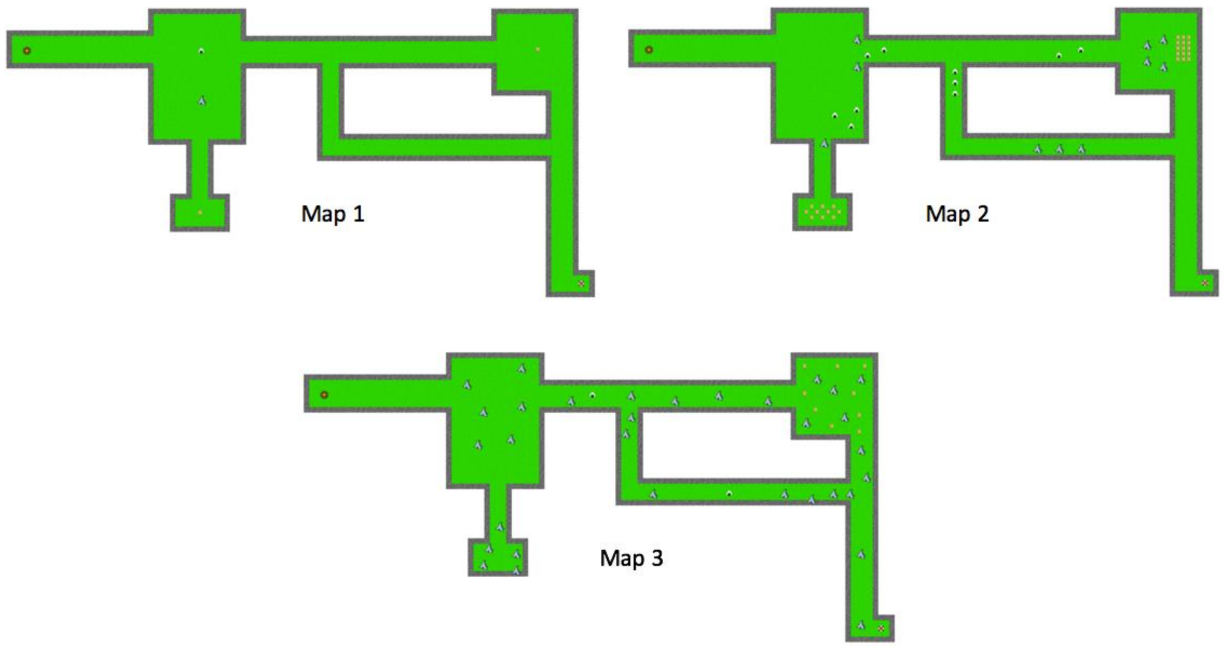


Figure 3: The three maps used for the collection of data. As can be seen, all the maps had the same topology yet the location and amount of objects differed between them.

The several input values and the output value were collected with a frequency of 8Hz. To tackle the sequential nature of the data, we decided to translate the input and output into overlapping slices of variable length (for example, one second), and using the variation of the values within that slice of time to train the model instead of the absolute values themselves. An exception were the input values related to the time elapsed since an event, which were also fed to the model in their absolute form as to allow the model to be aware of long periods of time where a given even didn't occur, for example, being aware that the player hasn't seen an enemy in over a minute.

The output slices were further classified as "increasing" or "decreasing/stable", transforming a prediction problem into a classification one. The absolute values for the emotional dimensions varied greatly between different users and didn't provide much information by themselves. To know if an emotional dimension was increasing or not was, however, a valuable information.

We then discarded all traces where there was no change to the emotional dimension throughout the entire play-through. These were the only traces that were discarded, all others being used.

Lastly, there were considerably more instances of the "decreasing/stable" class than of the "increasing" class. As such, the training data required balancing. From several methods tried, balancing using random over sampling proved to give the best results.

After gathering and processing the data as described, we were now faced with a traditional binary classification problem. We experimented with several different machine learning algorithms, such as neural networks, decision trees and state vector machines. In the end, the random forests algorithm was the one that provided the best results.

A different predictive model had to be trained for each one of the PAD emotional dimensions. As such, we achieved a different accuracy for each dimension. For the Pleasure dimension, we were able to achieve an accuracy of 72.8%. For the Arousal dimension, we were able to obtain a slightly better 73.1% of accuracy. However, our approach fared considerably worse on predicting the Dominance dimension, which we were only able to predict with around 60% accuracy.

The code pertaining to this section along with instructions on how to use it can be found on the GitHub repository [5]. The instructions can also be found in Annex A1.

Links and References:

References:

[1] J. A. Russell and A. Mehrabian. Evidence for a three-factor theory of emotions. Journal of research in Personality, 11(3):273–294, 1977.

[2] R. Plutchik. A general psychoevolutionary theory of emotion. In Theories of emotion, pages 3–33. Elsevier, 1980.

[3] P. Lopes, G. N. Yannakakis, and A. Liapis. Ranktrace: Relative and unbounded affect annotation. In 2017 Seventh International Conference on Affective Computing and Intelligent Interaction (ACII), pages 158–163. IEEE, 2017.

[4] D. Melhart, A. Liapis, and G. N. Yannakakis. The effect game annotation (again) dataset. arXivpreprint arXiv:2104.02643, 2021.

Source code: [5] https://github.com/iv4xr-project/PAD_emotion_game

Expected Publications:

Pedro M. Fernandes, Manuel Lopes, Rui Prada, Learning Emotion from Continuously Annotated User Traces. Manuscript in preparation.

The Appraisal model of emotions for Automated PX testing

In WP4, we are also investigating the use of a cognitive appraisal model of emotions for automated agent-based player experience (PX) testing. The idea is explained more in our abstract paper [6]. To meet this aim, at the beginning, we developed a formal model of appraisal for event-based emotion. In particular, we discuss an event-based transition system to formalise relevant emotions using Ortony, Clore, & Collins (OCC) theory of emotions [7]. The model is integrated on top of iv4xr's tactical agent programming library, to create intelligent PX test agents, capable of appraising emotions in our first game case study called Labrecruits. The results are graphically shown e.g. as heat maps. Visualisation of the test agent's emotions would ultimately help game designers to produce contents that evoke a certain experience in players. The results of a level called Lab1 are shown in Figures 4 to 6. Technical details of the model and the integration to Aplib can be seen in the paper [8], published in EMAS@AMAS. The code of the model of emotion is accessible through the project's Github repository [9]. The prototype version of automated agent-based player experience (PX) testing along with the instruction is available on a separate repository [10] which keeps getting updated.

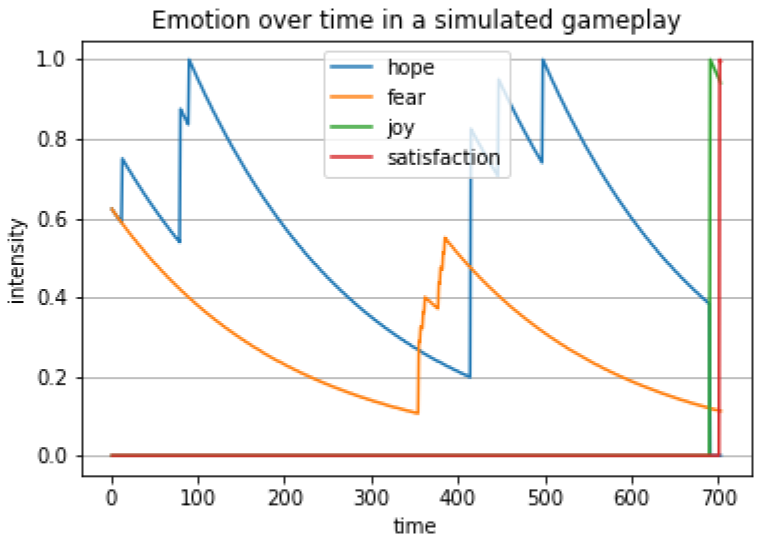


Figure 4: The emotions' timeline in Lab1 level setup.

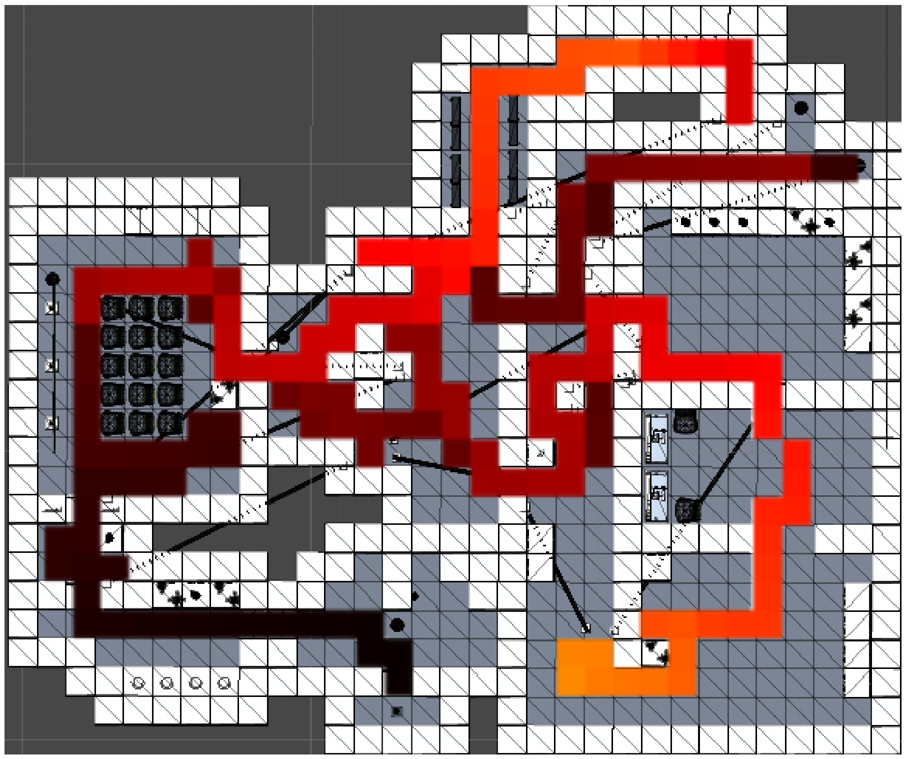


Figure 5: The heatmaps of negative triggered emotions

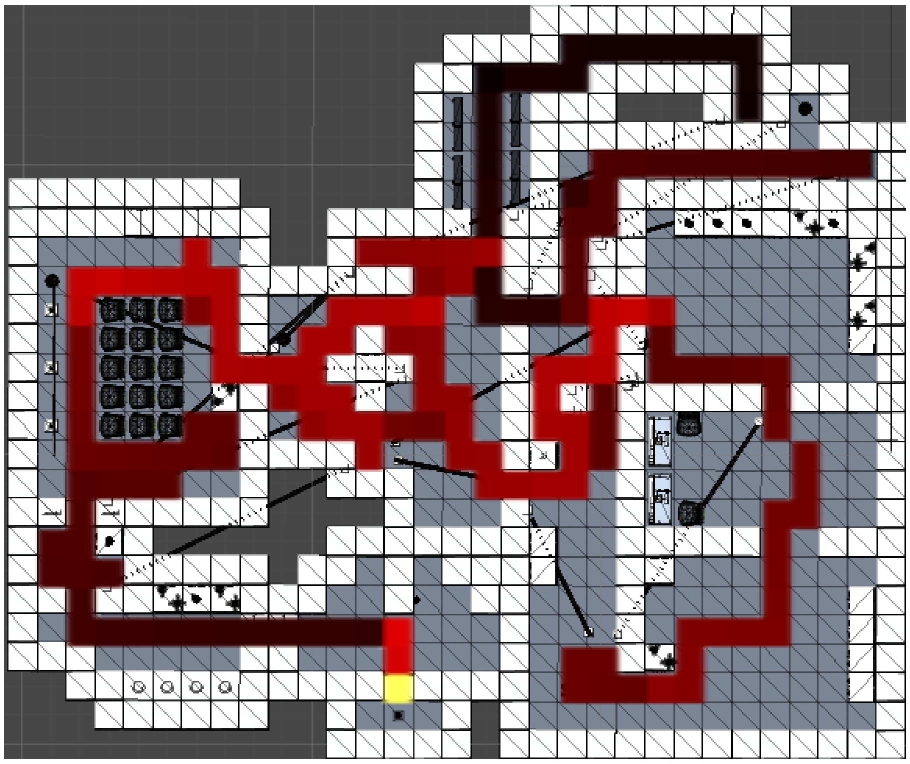


Figure 6: The heatmaps of positive triggered emotions

Afterward, we concatenated our working prototype with the model based testing (MBT) package from WP3, so now it is not only compatible with the last created version of iv4xr-core but also with the last version of iv4xr-MBT. Currently we are working on having automated testing of emotional requirements using model based testing. To do so, we are using extended finite state machines provided by MBT and we are formulating emotional requirements asked by game designers into linear temporal logic (LTL) formulas. We also use Buchi model checking existing in iv4xr-core and search based testing existing in iv4xr-MBT to generate test suites to be used for the emotional evaluation of the game-level under test against the given set of LTL emotional queries. This would create a whole pipeline for automated PX testing. The pipeline is almost completed, with only the LTL emotional evaluation part still being under development. The pipeline can be used along with iv4xr-MBT and iv4xr-core to create a test suite e.g. using buchi model checker in iv4xr-core and then using our model of emotions, you can get a visualised result of emotions for the level under test. Currently, the PX testing package uses a random level generator in WP3 to have any desired level for the test. The source code exists in the github repository in the branch called "pxtesingfr" [10]. To work with it, it is needed to have iv4xr-core [11], Lab Recruits iv4xr-demo project [12] and iv4xr-MBT [13].

Links and References

[6] Ansari, S. G. (2020, October). Toward automated assessment of user experience in extended reality. In 2020 IEEE 13th international conference on software testing, validation and verification (ICST) (pp. 430-432). IEEE.

[7] Ortony, A., Clore, G., Collins, A.: The cognitive structure of emotions. cam (bridge university press. Cambridge, England (1988)

[8] Ansari, S. G., Prasetya, I. S. W. B., Dastani, M., Dignum, F., & Keller, G. (2021). An Appraisal Transition System for Event-driven Emotions in Agent-based Player Experience Testing. Accepted in EMAS@AMAS, Springer (in press) arXiv preprint arXiv:2105.05589.

[9] https://github.com/iv4xr-project/jocc

[10] https://github.com/iv4xr-project/occ4pxtesting

[11]https://github.com/iv4xr-project/aplib

[12] https://github.com/iv4xr-project/iv4xrDemo

[13] https://github.com/iv4xr-project/iv4xr-mbt

Expected Publications:

Ansari, S. G., Prasetya, I. S. W. B., Dastani, M., Dignum, F., & Keller, G. . Automated testing of emotional requirements of a game level using model based testing, Manuscript in preparation.

GENERATING BEHAVIOUR FOR UX TESTING

As part of WP4, we are also exploring how to ensure the agent's behaviour is as "human-like" as possible, something which is necessary to ensure that the obtained UX measurements are reliable. An agent that constantly repeats a single action that doesn't alter the state of the environment might report a bad UX, but that doesn't necessarily mean the environment being tested is at fault. UX testing agents will be required to behave like users or else any obtained measurements of UX might prove meaningless. Generating behaviour and measuring UX will be two faces of the same coin and both will need to work together if truly automated UX testing is to be achieved. As such, we have decided to explore ways in which "human-like" or "persona-like" behaviour can be generated.

The behavioural generator of our UX testing agents should satisfy the following criteria:

1. Allow the agent to achieve the goals that users strive to achieve when using the system under test.
2. Pursue those goals in a way that resembles the behaviour of individual users or clusters of users.
3. Provide variety, allowing for the agents to behave in different ways when presented with the exact same scenario, just like different users often behave differently in the same situation.

Each of this criteria presents its own set of problems. Considering the "Flower Hunter" game presented in the previous section, to tackle criteria 1, we must endow the agent with the ability to navigate the map, fight enemies, and search for items. To tackle criteria 2, we have to ensure that the way in which the agent solves criteria 1 is similar to the way real users do it. Finally, our solution must allow for different traversals of the exact same level, all the while still satisfying criteria 1 and 2.

To satisfy criteria 1 for the "Flower Hunter" game, we needed to endow the agent with the ability to navigate a level. To do so, the agent needs to have an internal map of the level. The work from

WP3 already provides map navigation, but having criteria 2 already in mind, we decided it would be best if the agent was able to build the map as it traverses the level as opposed to having access to the totality of the level map from the beginning. An agent that knows from the beginning where all items are will not have an incentive to explore and will head straight to where it desires, something that only the luckiest of players will be able to do. This isn't to say there aren't situations where giving the full map won't be a solution as well, as for example to simulate a player that is playing a level for a second time and already knows where things are located. But this could also be simulated by having the agent play the level twice while saving the map from one play-through to the other. We might even have the agent do this several times if it keeps dying before reaching the end goal, something that real users often do in difficult games.

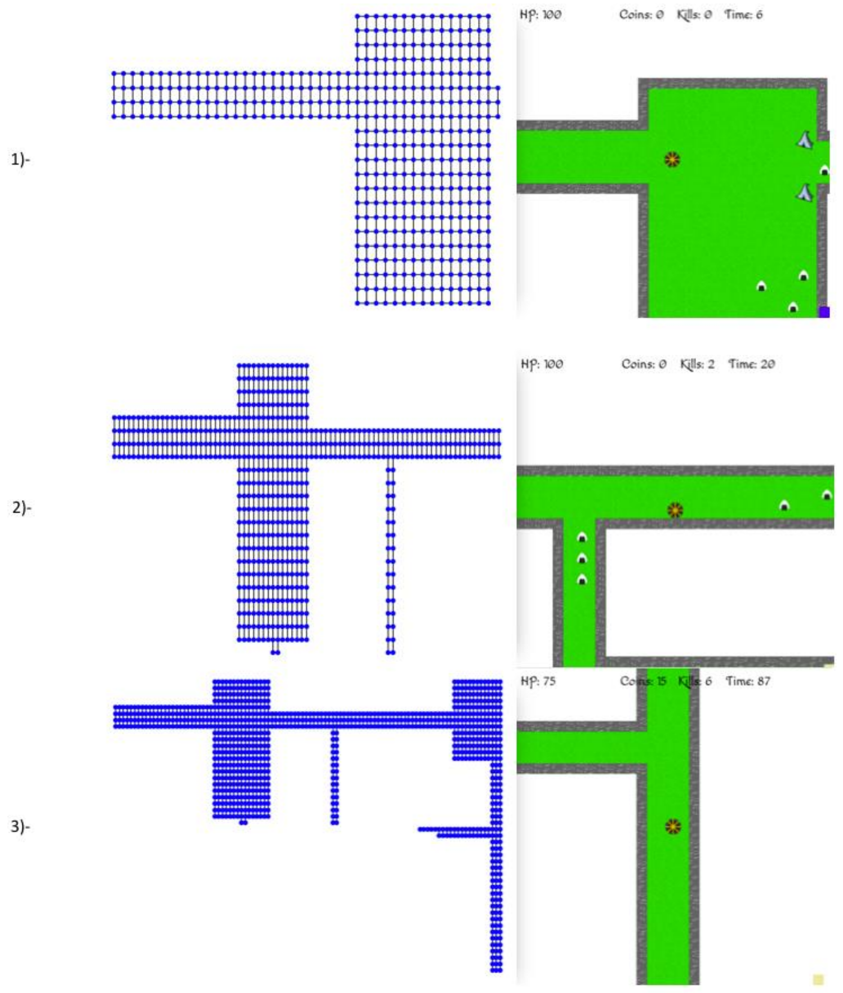


Figure 7: Sequence of 3 moments of the agent's traversal of Map 2 (Fig. 3). On the left side, the generated internal navigational graph of the agent. On the right side, the agent's position in the "Flower Hunter" game. As the agent traverses the level, it expands its navigational graph, as can be seen in the

evolution from 1) to 2) to finally 3). In all of the shown navigational maps, there are still areas of the map left unexplored given that the agent hasn't yet visited them.

Given that the objective of the developed agents was to test the game, we decided to build the map in a way that would most easily explore and detect errors in the map generation itself. We did so by using the game engine's collision detection. We created a square object in the game of a set dimension. The dimension of this square corresponded to the granularity of the map we would generate. We then iterated this square over all non-overlapping positions on the field of view of the agent, checking at each iteration if the square collided with any object. If no collision is detected, the position of the square is added to a "free positions graph". All adjacent nodes of this graph are connected. In this way, the agent can run classical path finding algorithms on the graph, like Dijkstra's algorithm, in order to navigate the level. The agent navigating the map and the generated internal navigational graph can be seen on Fig. 7.
Having found a solution to the problem of navigating the map, it was now trivial to make the agent move towards objects and fight enemies. But when should the agent do these things? When should the agent fight enemies? When should it collect coins or rice cakes instead? The final objective of the game was finding the flower, but there were many sub-objectives in the game that could be pursued as well. A player might decide to kill every enemy in the game or to ignore enemies and focus on collecting coins instead. This choice between what sub-objective to pursue is one of the things that makes different users behave differently. We therefore decided to emulate this by defining a set of parameters that would encode the behaviour of the agent and the preferences the agent had regarding the many possible objectives in the "Flower Hunter" game. The possible objectives were: exploring; reaching the flower; killing enemies; collecting coins; and collecting rice cakes. Having such objectives in mind, the parameters used were the following:

1. $ExploringPreference$
2. $ReachingFlowerPreference$
3. $KillingEnemiesPreference$
4. $CollectingCoinsPreference$
5. $CollectingRiceCakesPreference$
6. $Randomness$
7. $HealthHunger$

Each of these parameters was given a value between 0 and 10. Parameters 1 to 5 encoded the priority of the several possible goals that the agent could pursue. Parameters 6 and 7 were different. Parameter 6, Randomness, controlled whether the agent was deterministic or stochastic. A Randomness of 0 meant the agent was fully deterministic on choosing its priorities according to the parameters. A value of 10 meant the agent chose its priorities at random, regardless of the values of the other parameters. Values between 1 and 9 meant the agent chose it's priorities in a probabilistic way, as is detailed in Equation 1.7. Parameter 7 was a conditional parameter. Rice cakes provided health to the player, so we wanted to have a way to allow the agent to only gather rice cakes when its health was getting low. Parameter 7 translates how much the health value influences the interest of the agent on gathering rice cakes. Other such conditional parameters could be added and are likely to be added in future versions of the agents. The probability value of the agent pursuing each of the possible goals was given by:

$$V(Exploring) = ExploringPreference \times \mathbf{E} \tag{1.1}$$

$$V(ReachingFlower) = \frac{ReachingFlowerPreference}{distance(flower)} + ReachingFlowerPreference \times \mathbf{E} \tag{1.2}$$

$$V(KillingEnemies) = \frac{KillingEnemiesPreference}{distance(enemy)} \tag{1.3}$$

$$V(CollectingCoins) = \frac{CollectingCoinPreference}{distance(enemy)} \tag{1.4}$$

$$V(CollectingRiceCakes) = \frac{CollectingRiceCakesPreference}{distance(cake)} + \frac{HealthHunger \times \frac{\mathbf{MaxH-H}}{\mathbf{MaxH}}}{distance(cake)} \tag{1.5}$$

with *distance(x)* returning the distance to the nearest object of type *x*, **MaxH** being the maximum health of the player, **H** the current health of the player and:

$$\mathbf{E} = \begin{cases} 0, & \text{if the world has been fully explored} \\ 1, & \text{otherwise} \end{cases} \tag{1.6}$$

We can then define the probabilistic weight of the agent pursuing any one of the five possible goals, *P(goal)*, as:

$$P(goal) = (1 - Randomness) \times (V(goal) \times (IsMax(V(goal)) + Randomness)) \tag{1.7}$$

where:

$$IsMax(V(goal)) = \begin{cases} 1, & \text{if } (V(goal) \geq V(goal_i))\forall goal_i \neq goal \\ 0, & \text{otherwise} \end{cases} \tag{1.8}$$

In this way, we have defined the priorities of the agent based on parameters. Having also defined behavioural functions for the pursuit of each goal, we now have an agent that satisfies both behavioural criteria 1 and 3. We are currently working on using this work to also satisfy criteria 2, as will be discussed on Section 4 of this document.

The code pertaining to this section along with instructions on how to use it can be found on the GitHub repository [14]. The instructions can also be found in Annex A1.

Links and References:

Source code: [14] https://github.com/iv4xr-project/PAD_emotion_game

Expected Publications:

Pedro M. Fernandes, Manuel Lopes, Rui Prada, Generating Behaviour for Testing UX. Manuscript in preparation.

CREATING USER PROFILES BASED ON TRACES

In addition to modelling components of UX, we need agents that are capable of interacting with the system under test in order to test it. Moreover, for UX testing, we require that such agents behave in “human-like” ways.

We have shown in Section 3.2 a way of generating such behaviour based on parameterized agents. In this section, we will present an approach based on learning from observed behaviour, more precisely player traces. This learning approach is defined as apprenticeship learning, and in order to perform it we used the Inverse Reinforcement Learning (IRL) formulation. IRL as is originally defined in [15] and as the name implies, is to reverse the goal of standard Reinforcement Learning (RL) problems. In the standard RL, the goal is to generate policies, in IRL the goal is flipped meaning we intend to generate a reward from a given policy.

We described this problem 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 optimised

This is one of the principal components of our approach.

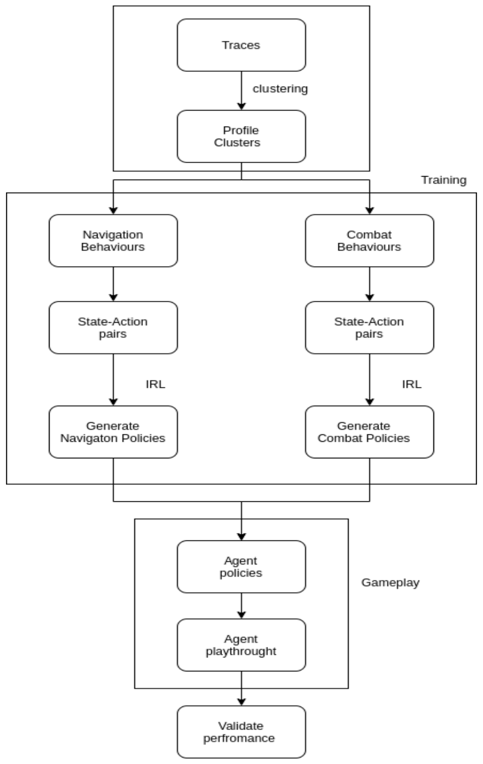


Fig 1. Diagram of the taken approach. First we take player traces and perform clustering to get the original profiles. Then with the clusters we perform IRL just to generate policies that illustrate said profiles. Finally we test them and compare the agent performance to that of original traces.

The environment that we decided to use, to train and test these agents, was the "Flower Hunter" game that was presented in section 3.1.1. We decided to use this environment since it had an associated dataset composed of several player traces which might reveal different player behaviours. Also since it is a part of the same project, the work done here can be used in other components of WP4. Finally, since this game was made in Python, it was easier to implement machine learning algorithms.

The dataset as mentioned before is composed of 264 traces that were collected from an experiment that involved 88 participants. Each trace is composed of 3 files: one where the player positions are recorded, another where all actions are recorded and finally one that stores the evolution of the input values (these can also be found in section 3.1.1).

Directly using all 264 traces on the IRL training led to unsatisfactory results, since the dataset isn't very homogeneous. We have different players (88 in this case) which might have different inner goals and ways of playing. Using all 264 traces at the same time generated very confusing agent policies.

To overcome this problem we first needed to sort the available player profiles. To achieve this we decided to apply a clustering procedure in order to group the players into profile clusters according to their performance. In order to create the clusters, we took 4 inputs from the available list, that we understand to be the most relevant ones, such as:

- Distance to objective
- HP
- Coins gathered
- Kills

Since the traces come from different levels that have different element quantities (example, number of enemies, in one level there are only 2 enemies, and in other there are 30), we decided to normalise the inputs that are connected to these elements, in this case Coins gathered and Kills.

When talking about the clustering algorithm used, we tested 3 different algorithms that were based on different clustering models, more specifically k-means, expectation-maximisation (EM) and hierarchical. Then we evaluated those algorithms using 2 metrics: mean square error (MSE) and silhouette score. The EM algorithm presented the best results, so it was chosen for our approach. With this algorithm we obtained 10 possible clusters.

Regarding IRL, instead of using the original implementation as defined in [15], we opted for our approach to be based on [16], in a technique named Maximum Entropy(MaxEnt) IRL.

As the name implies, this approach is based on the principle of max entropy. This principle 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 constraints of our prior knowledge.

In the case of the IRL method introduced in [16], MaxEnt describes a method of matching feature expectations between observed paths and optimal paths for recovered reward functions. In other words, if we generate a policy that is considered optimal for the recovered reward function, then it is expected that on average paths, generated by this policy, are equal to those of the optimal policy for the true reward function.

Regarding the model used to describe the game environment, we decided to use two MDPs: one for the navigation, where each agent state corresponds to a valid position in the game map, and the action space corresponds to movement in the 2D axis (RIGHT, LEFT, UP and DOWN) and the empty action STAY. For the combat MDP, we considered a more abstract approach for the state space. Here each state is represented as a combination of two features, agent HP, and distance to the closest enemy. For this MDP the action space is composed of following actions: moving towards, and away from the enemy, attacking the enemy and the empty action STAY.

In our approach, the agent will change from navigation mode to combat mode when an enemy is closer to the agent than a certain threshold. Then it will change back to navigation if the enemy is defeated or farther away from the agent.

As for the results obtained with this work, we were capable of recreating some of the profiles described by the clusters, although we encountered some limitations regarding the agent navigation. In table 1 and 2 we show some of the best results that we obtained.

	level	Time	Coins collect	Enemies killed	HP	map coverage
Agent 1	1	28	1	0	100	21.57%
Cluster 1	1	34	1	0	100	22.72%

Table 1. The profile found in cluster 1 is of players that move straight towards the final objective. We can see that our agent collected the same number of coins, explored a similar percentage of the level, and took a similar time to reach the end. From these results we can see that the agent was successful in capturing this profile.

	level	Time	Coins collect	Enemies killed	HP	map coverage
Agent 5	3	26	0	8	0	22.51%
Cluster 5	3	26	0	2	0	17.62%

Table 2.The profile found in cluster 5 is of players that died right at the beginning of level 3. From these results we can see that the agent was successful in capturing this profile, since it died almost at the same time step, although our agent was more efficient during combat.

In terms of future work, there are still ways in which our implementation can yet be improved. One is to change the model used in the navigation mode, since the original one is based on level positions, making it impossible for the obtained rewards\policies to be used in levels with different position layouts. Another component that needs closer inspection and improvements is the cluster solution. Some profiles revealed within that solution were shown to be very similar between each other, which hints that a deeper cluster analysis is needed.

Links and References

[15] Ng, Andrew Y., and Stuart J. Russell. "Algorithms for inverse reinforcement learning." *Icml*. Vol. 1. 2000.

[16] Ziebart, Brian D., et al. "Maximum entropy inverse reinforcement learning." *Aaai*. Vol. 8. 2008.

DIFFICULTY ESTIMATION

The difficulty of a level, and more importantly, the progression of difficulty of a series of levels, can have a significant impact on the user experience and learnability of a game. As such, we are developing methods to rank a series of levels in terms of their difficulty.

We have decided to use different types of errors as a way to measure the difficulty of a level. The main rationale behind this approach is that a level can be considered more difficult than another if the same degree of errors to the perfect sequence of actions leads to a worse outcome. For example, if a random timing error introduced to a perfect gameplay leads to the agent failing to complete a level 70% of the time whereas it only leads to the agent's failure 30% of the time in another level, we consider that the first level is harder than the second, at least regarding timing related mistakes from the player.

We have implemented 3 different error generation methods and used them to order a number of levels of a platformer game in terms of difficulty. The methodology and results for this study can be found on Annex A3. The code along with instructions on how to use it can be found on the project's GitHub repository [17].

Links and References

Source code: [17] https://github.com/iv4xr-project/difficultysch

Automated Cognitive-Load Estimation

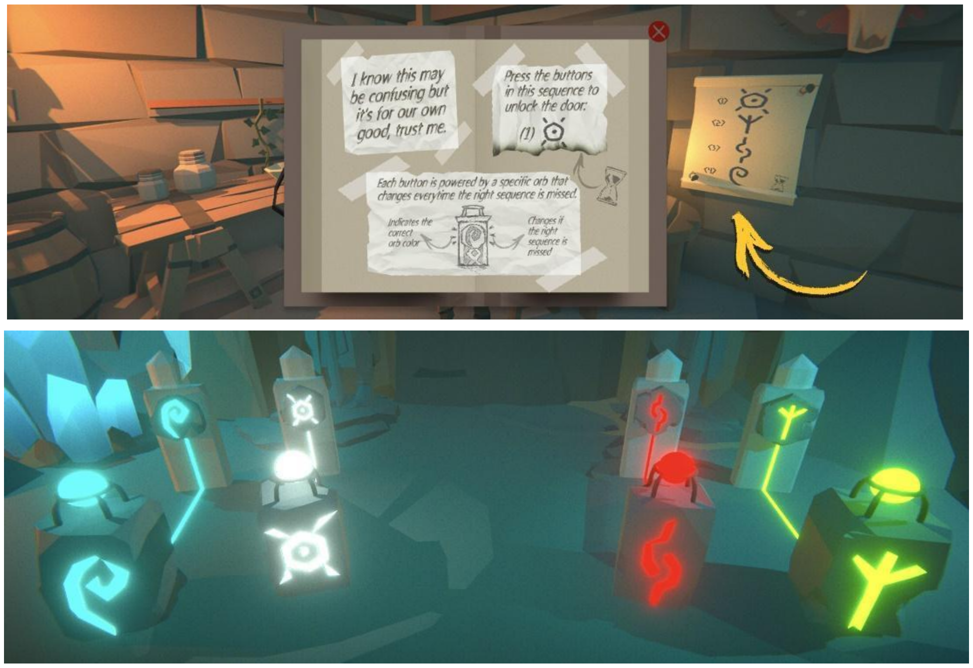
A significant part of XR user experience results from the interaction of each user with the system while solving a task in the environment. During interaction with an XR system, each user has different exchanges while navigating through the design space of the system, which will create distinct user experiences (UX). Aside from the emotions and social motivations, cognitive abilities also impact appraisal, and, indirectly, how the user navigates the space. Being able to understand how the interactions reflect these dimensions of the experience would help designing more personalised systems, resulting in better UX.

This work focuses on Cognitive Load, meaning the amount of information a person is conscientiously processing at a given moment. The cognitive load has a close relationship with attention mechanisms. We aim to create a toolset in the context of automatic play-testing that we can extend to other types of systems. The toolset, based on the TBRS (Time-Based Resource Sharing) [18] memory model, aims at providing a measure of the cognitive load (a percentage) that the user is expected to experience while going through a particular task in a specific context. Autonomous agents navigating and exploring a virtual environment need some parameterisation of what grabs the agents' attention (e.g., interacting with an object or dodging enemies' attacks). We call these "attention-grabbing events", measured in seconds (duration of the event). The toolset will provide a set of methods (API) that need to be added/called from the code of the software undergoing testing, each time an attention-grabbing event occurs.

After finalising a task, the toolset will compute an estimate for the cognitive load experienced by the user, based on the sum of the attention-grabbing events, and the total duration of the task. We can integrate this value into automated testing procedures by using assertions in the code that check whether the estimated value of the cognitive load is within a specific desired range. If not, the toolset will be able to present a short report to identify possible problems with the current implementation (based on the data gathered) and shed some light on the direction to take in future development.

To evaluate our model, we created the game "Way-out" (Figure 8), in which the player is escaping from a small underground complex. We designed the game to allow for the parameterisation and control of several attention-grabbing events, and the manipulation of several dimensions of the experience, such as the time required to navigate through the game, the complexity of the task based on the number of interactions required to overcome them, as well as the number of items a player needs to keep in mind to solve the different puzzles. By comparing the reported cognitive load of the participants to the value computed by the TBRS theoretical model, we aim at evaluating whether this model is an appropriate predictor of the cognitive load reported by the players for the whole experience.

We conducted a user study which showed that the attention grabbing events are correlated with higher reported cognitive load. However, our time manipulation was not successful as the game is self-paced.


I know this may be confusing but it's for our own good, trust me.

Press the buttons in this sequence to unlock the door:

(1)

Each button is powered by a specific orb that changes everytime the right sequence is missed.

Indicates the correct orb color
Changes if the right sequence is missed

Figure 8: Screen shots from the 'Way out' game

During the second year, we are working on a plugin that allows for a more fine-grained measure of cognitive load. The plugin introduces a secondary task in the game. In the secondary task, players need to press a predefined button every time a certain event happens, for example, when a red dot appears on the screen. Increased reaction times to the secondary task, i.e. longer time intervals between the stimuli and the response, indicate higher cognitive load. This type of task has the advantage of allowing us to pinpoint areas in the game where the user experiences higher cognitive load and we can relate them to the concentration of attention grabbing events. We will start by using the same game, "Way-Out" because we already have information on the overall cognitive load imposed by the levels we tested and information on the attention grabbing events. We are planning a new user study to evaluate our model.

Because we are still testing our model and developing the plugin, we have the code available on GitHub, but we do not yet have a user-friendly version. For delivery 4.3 we aim to have a tutorial for developers/designers that want to use the plugin to check the Cognitive load imposed on users by their systems.

We plan to submit a journal article detailing our first user study in the beginning of 2022 and will work to publish the results of our new study.

Links and References

References:

[18] Barrouillet, P., & Camos, V. (2007). The time-based resource-sharing model of working memory. *The cognitive neuroscience of working memory*, *455*, 59-80.

Project Github:

https://github.com/albertoramos1997/WayOut

Video(s):

Vídeo Way Out (Lever Puzzle): The video shows the four versions of the "lever puzzle". In this puzzle, the player needs to collect the missing levers, add them to a machine, and activate the correct levers to open the door to the next room. The video also shows the inventory (backpack) and the notebook, where players can store hints about the different puzzles.

https://www.youtube.com/watch?v=6LSi81yiB28&t=18s&ab_channel=AlbertoRamos

Vídeo Way Out (Orb Puzzle): The video shows the four versions of the “orb puzzle”, which is the final puzzle of the game. Throughout the rooms, the player finds different orbs in stands. Each orb has a different colour and each stand has a symbol. The player has four stands with symbols on the final room, and they need to match the symbols to the colours. Once all orbs are in the correct stand, the player needs to activate the buttons (by pressing the symbols) in a predefined order to win the game.

https://www.youtube.com/watch?v=8ge565wPE9I&t=21s&ab_channel=AlbertoRamos

Expected Publications:

[19] Ramos, A., Couto, M., Martinho, C. (2021). *Assessing Players’ Cognitive Load in Games*. Manuscript in preparation to be submitted to a journal.

VERIFICATION OF INTERACTION PROPERTIES

On D4.1, we presented our initial work on the verification of interaction properties between two players or agents. We have further consolidated this work by analysing the problem from a different perspective.

Our goal is to verify if certain behaviours can occur in a specific game level and if so, to measure the level's susceptibility for those behaviours. To simplify, we decided to start with the collaboration behaviour. In other words, we are trying to evaluate how collaborative a game level is.

Our approach to address the problem was to create three types of agents and use their learned behaviours to evaluate a game level. Each level has a set of buttons, some of them unlock doors to other rooms, while others are target buttons. The goal of each level is to press all target buttons. The scenarios are loaded from a .txt file.

The code for this project can be found on the project's GitHub repository [20]. The GridWorld class sets up the environment for the agents to learn their policies or to simply execute learnt policies. Therefore, it includes methods such as:

- Reset or RandomReset
- CheckGoal
- GetState (GetStateFullObservability, GetStatePartialObservability)
- Learn (LearnDecentralized, LearnCentralized)
- Step (StepSingleAgent, StepDecentralized, StepCentralized)
- EvalAgents (EvalAgentsDecentralized, EvalAgentsCentralized)
- Render
- WriteAgentsQtableToFile
- LoadAgentsQtableFromFile

As previously mentioned, three types of agents were developed: the centralised, the decentralised and the single agents. The following characteristics were common to all agents:

- Action space is [UP, DOWN, LEFT, RIGHT, PRESS, NOTHING].
- Action selection is done with the Egreedy algorithm and when choosing the current best action for a certain state, ties are solved randomly.
  - E (or epsilon) decays linearly from 1 to MIN_EPSILON (set as 0.05) during 70% of the episodes. The last 30% of the episodes, E remains at 0.05.

- Rewards:

  - 100 for reaching the goal
  - -0.4 when choosing NOTHING
  - -1 for any other action
  - Notes: the reward of doing NOTHING constrains the time it takes for the agents to learn.
- Learning algorithm is DynaQ with 10 planning steps.

The **centralised agent** is a single entity controlling the actions and knowledge of two players. The state is a string with [pos_x_a0] + [pos_y_a0] + [pos_x_a1] + [pos_y_a1] + [state of all the buttons]. Because it maps the position of the two players (and all the buttons) it is accessible with a method called GetStateFullObservability.
The possible action is a numeric value mapping the action of each player. For instance 0 represents the player0 doing UP and player1 doing UP, while 1 represents the player0 doing UP and player1 doing DOWN.
The rewards were doubled to have fair comparisons with other agents.
The **decentralised agents** are two distinct instances, each representing one player.
The state is a string with [pos_x] + [pos_y] + [state of all the buttons]. Because it maps the position of only one player, it is accessible with a method called GetStatePartialObservability.
These agents learn in a decentralised way but they learn at the same time in the same environment. Therefore, what happens is that the agents actually learn to cooperate if that gives them a higher reward.
The **single agents** are independent of one another as they learn alone in the environment and therefore, the learnt policies correspond to an individualist behaviour.
The learning method is in the Agent class file: LearnDynaQ. When implementing this type of agent, a problem occurred while evaluating two simultaneous agents of this type in the same environment. As soon as one of the agents opened one door, the other no longer knew what to do as it had never seen that state. To address this issue, at learning-time, instead of resetting the agents to always start in the same state, we used the method RandomReset. This method, 30% of the time, resets the environment to a random state (including position and button states).

Running Instructions: The main.py can be executed as follows:

- main.py -learn [AGENT_FLAG] [SCENARIO]
  - In this mode, the agents learn their policies for that specific scenario and, in the end, the Qtables are saved to a .txt file in the /policies/ folder.

- main.py -run [AGENT_FLAG] [SCENARIO]
    - In this mode, the agents load the Qtables saved in the /policies/ folder (if any) and execute the learnt actions step by step, while the console log visually displays their behavior.
- main.py -compare [AGENT_FLAG] [AGENT_FLAG] [SCENARIO]
    - In this mode, you will get some evaluation measures comparing the two types of agents (given in the execution parameters).

[AGENT_FLAG] can either be -centralised, -decentralised, or -singleagents
[SCENARIO] can either be -s1, -s2, or -s3

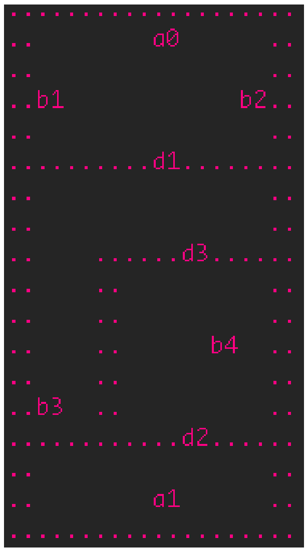
These following measures were used for the comparison of the agent's behaviours.
Reward difference (Δ reward): Absolute difference between the accumulated rewards of the two types of agents. The accumulated rewards are returned by the EvalAgents method. For the centralised agents, the output of their behaviour is joint in one accumulated reward for the two players. However, for both the decentralised and single agents, the output of their behaviour is the accumulated reward for each player, which is then summed. For this reason, the rewards of the centralised agents are doubled or "as summed".
Hitmap per Player Difference (Δ hitmap / player): Average difference between the behaviour trace of a player for each type of agent (X and Y). The method CompareHitmapsPerAgent receives the size of the grid, and the two traces of each type of agent. A trace is a list of positions, in which each two elements of the list are the position of the two players (agent0 followed by the position of agent1). Four zeros matrices with the size of the grid are created, one for each player (a0 and a1) of each type of agent (X and Y). While iterating over the traces, 1 unit is summed in the corresponding matrix cell each time a certain player of a certain type passed in that position of the grid. Then, the norm of the difference is calculated (numpy.linalg.norm) between each player of type X and the same player of type Y. Then the method returns the average of the two players.
Because the hitmap difference might be substantially different for one player and not for the other, the hitmap difference for each player (a0 and a1) is written in the results table. The Δ hitmap per agent is the average between Δ hitmap a0 and Δ hitmap a1.
Joint Hitmap Difference (Δ hitmap joint): Average difference between the behaviour trace of all the players for each type of agent (X and Y). The method CompareHitmaps does not compare for each player and instead creates a single hit map of where all players of a certain type have been.

Policies Difference (Δ policies): The policy matrix is binary as the agents have a deterministic behaviour. Instead of having a policy matrix for each player (as done for the hitmaps per player), here a single joint policy matrix was created for the two players (a0 and a1) as the centralised agents do by default. While for the centralised agents the policy matrix is obvious to get from the QTable (method GetPolicy), for both the decentralised and single agents it needs to be inferred/transformed. The method CreateJointPolicy does that, i.e., converts the two Qtables of the two players that have an action space of [UP,...,NOTHING] into a single Qtable (and therefore policy matrix) where the action space is [UP-UP,UP-DOWN,...,NOTHING-NOTHING].

*	C vs D	C vs S	D vs S
Scenario 1	Δ reward - 13.4 Δ hitmap a0 - 5.9 Δ hitmap a1 - 5.7 Δ hitmap / player - 5.8 Δ hitmap joint - 8.2 Δ policies - 89.9	Δ reward - 36.6 Δ hitmap a0 - 10.2 Δ hitmap a1 - 43.3 Δ hitmap / player - 26.7 Δ hitmap joint - 44.3 Δ policies - 92.6	Δ reward - 23.2 Δ hitmap a0 - 8.9 Δ hitmap a1 - 43.7 Δ hitmap / player - 26.3 Δ hitmap joint - 44.4 Δ policies - 89.2
Scenario 2	Δ reward - 1.4 Δ hitmap a0 - 6.1 Δ hitmap a1 - 5.6 Δ hitmap / player - 5.8 Δ hitmap joint - 8.2 Δ policies - 179.5	Δ reward - 7.0 Δ hitmap a0 - 3.0 Δ hitmap a1 - 3.0 Δ hitmap / player - 3.0 Δ hitmap joint - 4.2 Δ policies - 229.5	Δ reward - 8.4 Δ hitmap a0 - 4.5 Δ hitmap a1 - 5.5 Δ hitmap / player - 5.0 Δ hitmap joint - 7.1 Δ policies - 171.1
Scenario 3	Δ reward - 1.8 Δ hitmap a0 - 4.3 Δ hitmap a1 - 10.0 Δ hitmap / player - 7.2 Δ hitmap joint - 11.0 Δ policies - 183.1	Δ reward - 1.8 Δ hitmap a0 - 4.1 Δ hitmap a1 - 9.0 Δ hitmap / player - 6.6 Δ hitmap joint - 9.8 Δ policies - 181.2	Δ reward - 0 Δ hitmap a0 - 1.4 Δ hitmap a1 - 7.7 Δ hitmap / player - 4.6 Δ hitmap joint - 7.9 Δ policies - 145.4

Table 1: Table showing the comparison between the three agents. *C - centralised agents, D - decentralised agents, S - single agents

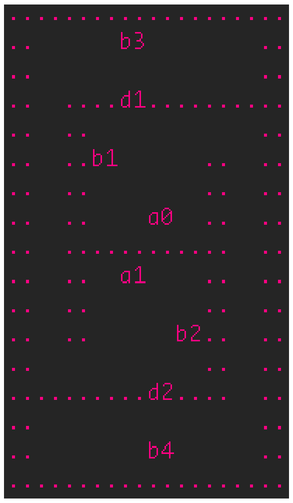
Scenario 1: The first scenario is interesting because a1 is trapped and the only way out is if a0 presses the button b2 (to open the door d2). Player a0 is capable of solving the level alone if he presses b1 and then b3, which makes it possible for him alone to press the one and only goal/target button b4.



The behaviour learnt by the *centralised agents (C)* corresponds to the collaborative behaviour, i.e., a0 presses b2 and a1 goes directly to the target button b4 (13 steps). What the *single agents (S)* have learned to do, because they train alone in the environment, is precisely the opposite, i.e., a1 does nothing while a0 presses b1, b3 and finally b4 (43 steps). The *decentralised agents (D)*, as mentioned before, have also learned cooperative behaviours as they are advantageous. However, because they don't fully observe the environment (i.e., they don't know the position of the other player), they take slightly more steps (20 steps) than the centralised agents. Specifically, in the centralised mode, a0 does not move much after pressing b2. Conversely, in the decentralised mode, a0 presses b1 after pressing b2. To sum up, C and D use both a collaborative strategy but D is slightly less efficient, and S uses a totally different strategy (that can be considered the least collaborative).

Based on this subjective description of their behaviour (which you can also check by running *main.py -run ...*), it was expected that numeric comparisons should highlight (1) higher differences between C vs S and D vs S than between C vs D. Moreover, it would be expected that (2) C vs S present a higher difference than D vs S. All the measures support the first expected result. However, the only measure that mirrors the second expected result is the comparison of rewards (Table 1).

Scenario 2: In the second scenario, no player is blocked in a room and there are two target buttons. Each of the target buttons is much closer to one of the players than to the other. As the

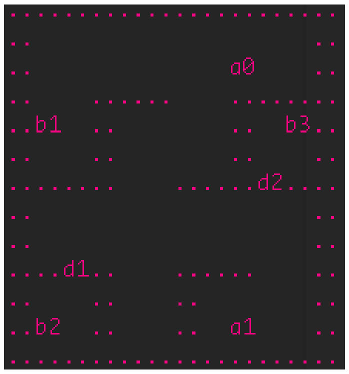


target buttons are two and they are equally accessible to the players, to solve this level players can act in parallel, such as in two independent sub-tasks.

The behaviour of the three types of agents is generically the same, i.e., a0 presses b1 and then b3, while a1 presses b2 and then b4. They slightly differ in the number of steps (C - 18 steps, D - 19 steps, S - 13 steps), but the expected numeric differences should be negligible (or zero by increasing the training time).

All the measures support that the behaviours of the three types of agent act similarly (Table 1).

Scenario 3: In the third scenario, there is only one target button and cooperation is advantageous to reach the goal slightly faster. In three modes, the learned behaviour generically corresponds to a0 pressing b1, a1 pressing b2 and then a0 is always the fastest to reach the target button b3. In both C and D, a0 is going towards b1, at the same time that a1 is going towards b2. However, in C mode, a1 stays still after pressing b2, while a0 is going towards the target button. Conversely, in D mode, because the agents do not know the position of the other player, they both go towards the target button (34 steps). In the S mode, both players start the level going towards b1, which is visible because a1 starts moving upwards in the central corridor before a0 reaches b1. As soon as d1 is open, a1 goes towards b2 and then both players go towards b3 to finish the level (34 steps).



Overall all measures suggest the three behaviours were similar and close (Table 1).

Links and references

Project Github:

[20] Github repository: https://github.com/iv4xr-project/rl-behaviors-verification

VALIDATING THE PLOT OF INTERACTIVE NARRATIVE GAMES

Many games and simulations include interactive dialogue. When the choices made during such dialogues impact the environment and possible future choices, then it is paramount that these interactive narratives are well tested to ensure a positive UX.

As such, we have focused on interactive narrative games to develop a model consisting of a set of metrics for testing interactive dialogues. Using this model, we developed a prototype for the Story Validator tool. This tool allows game writers to experiment with different hypotheses and narrative properties in order to identify inconsistencies in the authored narrative and predict the output of different playthroughs with visual representation support. We conducted a series of user tests using the Story Validator, to investigate whether the tool adequately helps users identify problems that appear in the game's story. The results showed that the tool enables content creators to easily test their stories, setting our model as a good step towards automated verification for assistance of authoring interactive narratives.

More details, including the methodology and results from this study can be found on Annex A4. The code can be found on the project's GitHub repository [21].

Links and references

Project Github:

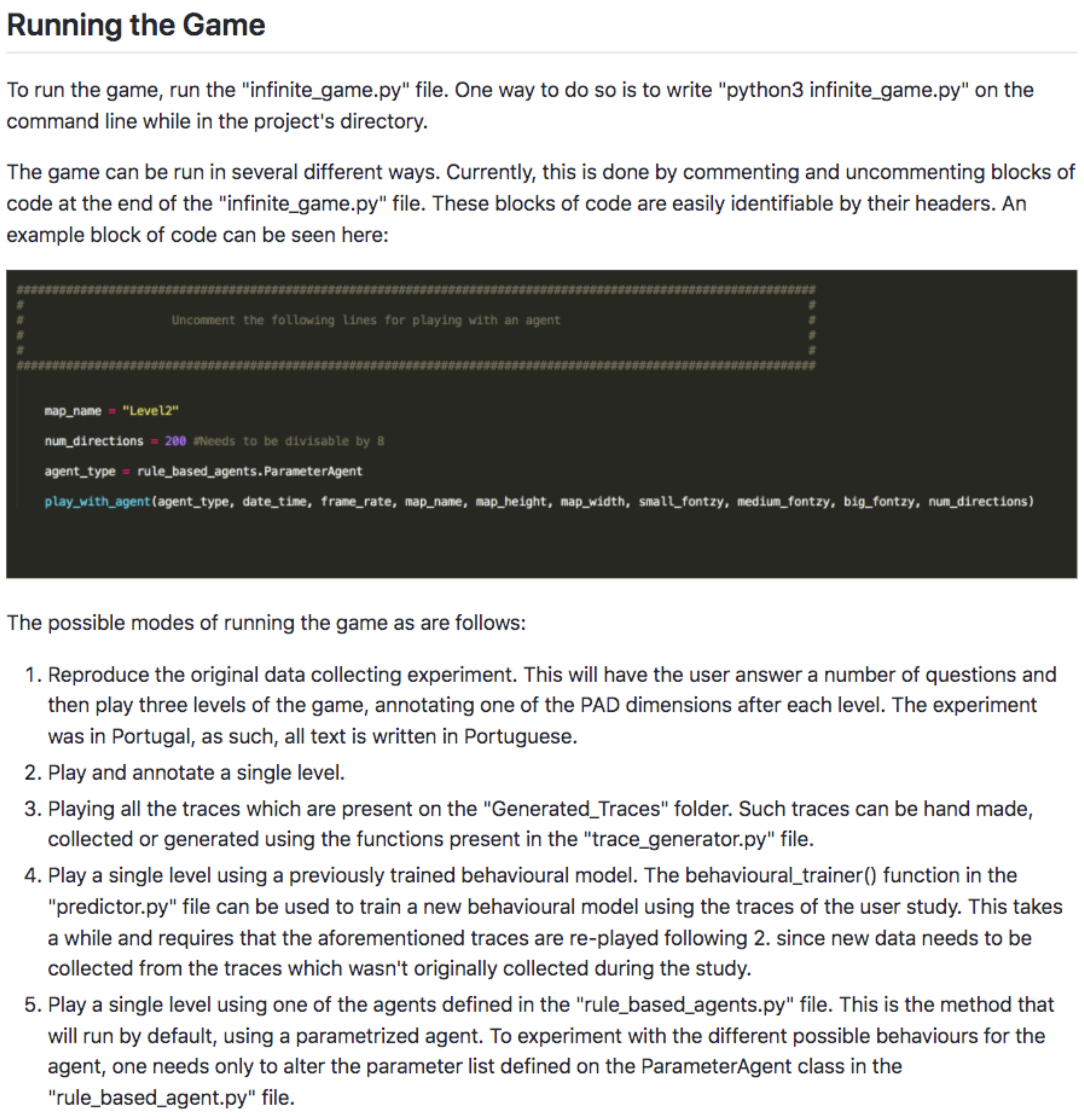
[21] Github repository: https://github.com/iv4xr-project/in-story-validator

CONCLUSIONS AND FUTURE WORK

In this period, we continued our work on developing methods to measure and evaluate various aspects of the experience of the user during gameplay. Some of this work was a continuation of what had already been shown in deliverable 4.1, whereas some of it is novel work. During the third period of the project, we will finally integrate all these different methods into a single UX testing module, which will be integrated with the framework. Designers and testers will then be able to choose which of the methods they desire to use in order to test the UX of their system. A paper is currently being written to clarify how the several components which we have explored are related and how they can be used to provide a picture of UX. Aside from the components hitherto presented, we are further exploring the generation of behaviour for multi-agent scenarios. Our focus is on generating relevant interaction traces that represent the different ways in which users might interact with one another.

ANNEXES

ANNEX A1



Running the Game

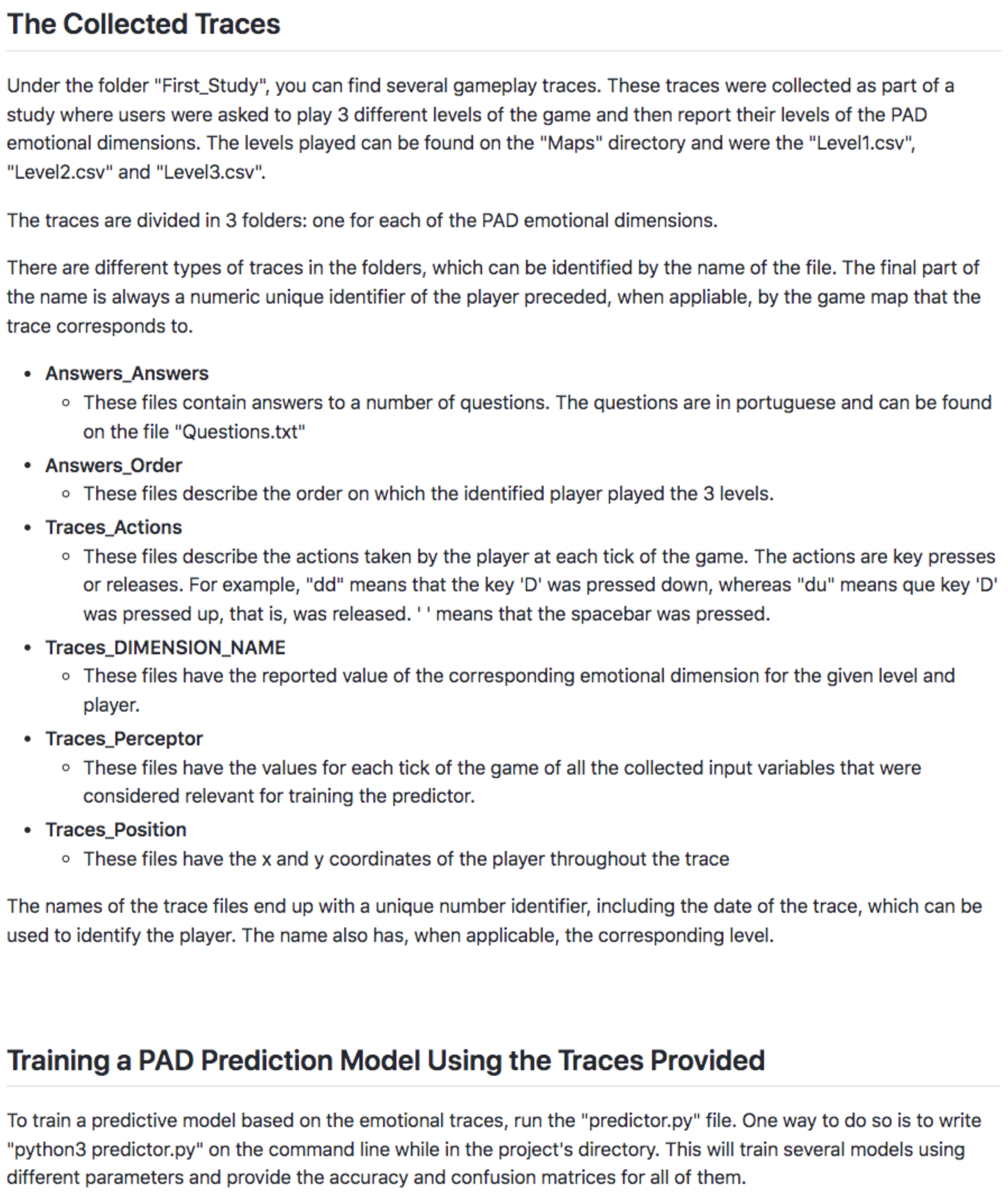
To run the game, run the "infinite_game.py" file. One way to do so is to write "python3 infinite_game.py" on the command line while in the project's directory.

The game can be run in several different ways. Currently, this is done by commenting and uncommenting blocks of code at the end of the "infinite_game.py" file. These blocks of code are easily identifiable by their headers. An example block of code can be seen here:

```
##############################################################################################################
#                                                                                                            #
#                     Uncomment the following lines for playing with an agent                                #
#                                                                                                            #
#                                                                                                            #
##############################################################################################################

    map_name = "Level2"

    num_directions = 200 #Needs to be divisable by 8

    agent_type = rule_based_agents.ParameterAgent

    play_with_agent(agent_type, date_time, frame_rate, map_name, map_height, map_width, small_fontzy, medium_fontzy, big_fontzy, num_directions)
```

The possible modes of running the game as are follows:

1. Reproduce the original data collecting experiment. This will have the user answer a number of questions and then play three levels of the game, annotating one of the PAD dimensions after each level. The experiment was in Portugal, as such, all text is written in Portuguese.
2. Play and annotate a single level.
3. Playing all the traces which are present on the "Generated_Traces" folder. Such traces can be hand made, collected or generated using the functions present in the "trace_generator.py" file.
4. Play a single level using a previously trained behavioural model. The behavioural_trainer() function in the "predictor.py" file can be used to train a new behavioural model using the traces of the user study. This takes a while and requires that the aforementioned traces are re-played following 2. since new data needs to be collected from the traces which wasn't originally collected during the study.
5. Play a single level using one of the agents defined in the "rule_based_agents.py" file. This is the method that will run by default, using a parametrized agent. To experiment with the different possible behaviours for the agent, one needs only to alter the parameter list defined on the ParameterAgent class in the "rule_based_agent.py" file.



The Collected Traces

Under the folder "First_Study", you can find several gameplay traces. These traces were collected as part of a study where users were asked to play 3 different levels of the game and then report their levels of the PAD emotional dimensions. The levels played can be found on the "Maps" directory and were the "Level1.csv", "Level2.csv" and "Level3.csv".

The traces are divided in 3 folders: one for each of the PAD emotional dimensions.

There are different types of traces in the folders, which can be identified by the name of the file. The final part of the name is always a numeric unique identifier of the player preceded, when appliable, by the game map that the trace corresponds to.

- **Answers_Answers**
  - These files contain answers to a number of questions. The questions are in portuguese and can be found on the file "Questions.txt"
- **Answers_Order**
  - These files describe the order on which the identified player played the 3 levels.
- **Traces_Actions**
  - These files describe the actions taken by the player at each tick of the game. The actions are key presses or releases. For example, "dd" means that the key 'D' was pressed down, whereas "du" means que key 'D' was pressed up, that is, was released. ' ' means that the spacebar was pressed.
- **Traces_DIMENSION_NAME**
  - These files have the reported value of the corresponding emotional dimension for the given level and player.
- **Traces_Perceptor**
  - These files have the values for each tick of the game of all the collected input variables that were considered relevant for training the predictor.
- **Traces_Position**
  - These files have the x and y coordinates of the player throughout the trace

The names of the trace files end up with a unique number identifier, including the date of the trace, which can be used to identify the player. The name also has, when applicable, the corresponding level.

Training a PAD Prediction Model Using the Traces Provided

To train a predictive model based on the emotional traces, run the "predictor.py" file. One way to do so is to write "python3 predictor.py" on the command line while in the project's directory. This will train several models using different parameters and provide the accuracy and confusion matrices for all of them.

ANNEX A2

An Appraisal Transition System for Event-driven Emotions in Agent-based Player Experience Testing [*]

Saba Gholizadeh Ansari[1] [0000−0002−7135−5605], I. S. W. B. Prasetya[1] [0000−0002−3421−4635], Mehdi Dastani[1] [0000−0002−4641−4087], Frank Dignum[2] [0000−0002−5103−8127], and Gabriele Keller[1] [0000−0003−1442−5387]

[1] Utrecht University, Utrecht, the Netherlands, s.gholizadehansari@uu.nl
[2] Umeå University, Umeå, Sweden

NOTE: This is a preprint of the article , accepted in 9th International Workshop on Engineering Multi-Agent Systems (EMAS2021), held as a part of 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS).

Abstract. Player experience (PX) evaluation has become a field of interest in the game industry. Several manual PX techniques have been introduced to assist developers to understand and evaluate the experience of players in computer games. However, automated testing of player experience still needs to be addressed. An automated player experience testing framework would allow designers to evaluate the PX requirements in the early development stages without the necessity of participating human players. In this paper, we propose an automated player experience testing approach by suggesting a formal model of event-based emotions. In particular, we discuss an event-based transition system to formalize relevant emotions using Ortony, Clore, & Collins (OCC) theory of emotions. A working prototype of the model is integrated on top of Aplib, a tactical agent programming library, to create intelligent PX test agents, capable of appraising emotions in a 3D game case study. The results are graphically shown e.g. as heat maps. Emotion visualisation of the test agent would ultimately help game designers in creating content that evokes a certain experience in players.

Keywords: automated player experience testing, emotional modeling of game player, formal model of emotion, intelligent agent, agent-based testing

1 Introduction

With the growing interest of industry and academia in assessing the quality in-use of a system, product or service, the term *User eXperience* (UX), which refers to quality characteristics related to internal and emotional state of a user, has emerged [19,22]. UX evaluations become essential for designers to predict how users would interact with a system. In the context of computer games, evaluating *player eXperience* (PX) plays an important role to design a well-received game according to players' preferences and expectations. PX has different dimensions such as flow [21], immersion [13] and enjoyment [8] which need to be addressed in a game design to evoke certain experience.

* This work is funded by EU H2020 Research and Innovation grant 856716, project iv4XR.

To assess the UX quality of a game, relatively novel UX evaluation methods such as questionnaire methods, psycho-physiological measurement and eye-tracking have been used [4,22,28]. Currently, PX testing techniques not only impose excessive hours of testing but they might also not be representative enough to cover all player types and their possible emotions towards the game. Despite some attempts towards automation, most of these techniques are either costly or still manually demanding [22,4,28]. Moreover, similar to UX evaluations in non-game applications, most of PX testing methods measure PX toward the end of the game development [2,4,28], so there is still a need for more efficient techniques to do these evaluations in *early stages* of game development. This allows PX problems to be addressed early during the development.

All of these factors led us to propose an automated approach for PX testing in computer games; the envisaged main use case is to assist designers in early development phases to develop their games more efficiently. To meet this aim, here, we proposes to employ a *computational model* of players to automatically assess PX properties of a computer game. Such a model is necessarily tied to cognition and emotion. Additionally, emotions that a player can feel under certain conditions would eventually affect their overall experience. We, therefore, suggest to deploy a well-known *theory of emotions* called **OCC** [17] to facilitate modeling players with respect to their emotions.

We present a formal model of the appraisal for OCC emotions using an event-based transition system to serve as the foundation of our automated PX testing approach. It deviates from existing formalization e.g. [1,10,26]; they have never been used in the software engineering (SE) domain. This might explain why these formal models have not been utilized for UX/PX testing. A more fundamental reason is that these models are given in the form of BDI[3] logic [15]. Although expressive, BDI logic is more a reasoning model rather than a computation model. In contrast, our formalization is given in terms of a transition system that directly specifies how to compute the emotional state. Having a transition system provides an opportunity for developers to simply deploy the model in their own systems, whereas a BDI-based formal model would also need a BDI reasoning engine before it can be used for computing. Furthermore, discrete transition systems have been used to do model checking in software for decades. This opens a way to express UX/PX properties in e.g. LTL or CTL [3] and verify them through model checking or model checking related techniques.

A prototype implementation of the formal model is also presented in this paper, along with a demonstration of what it can do on a small case study. The prototype of appraisal model is integrated with Aplib [20], a Java library for agent-based game testing, to create an **emotional test agent** that uses the OCC theory for emotional appraisal to assess PX requirements in games.

The paper is structured as follows: Section 2 introduces the OCC theory. Section 3 gives an overview of the proposed framework architecture as well as the role of appraisal in PX evaluations. Section 4 details the formal model of appraisal for event-based emotions. Section 5 explains the early results of the framework in a 3D case study. Section 6 discusses some related work and finally Section 7 concludes the paper and presents future work.

[3] Belief-Desire-Intention

2 OCC theory of emotion

Ortony, Clore, and Collins [17] presented a cognitive structure of emotions which characterizes 22 emotion types (e.g. joy, hope, disappointment, distress and fear). According to their 'OCC' theory, emotions are valenced reactions which can be turned on by outcome of events, outcome of agents' actions, or attributes of objects. Event-based emotions that are applicable to most game setups are highlighted in blue in Figure 1. We selected them to be the basis of our proposed event-based transition system for emotions in our PX testing framework (further explanation in Section 3.1). Each of the emotion types listed in Figure 1 is specified as described in [17].

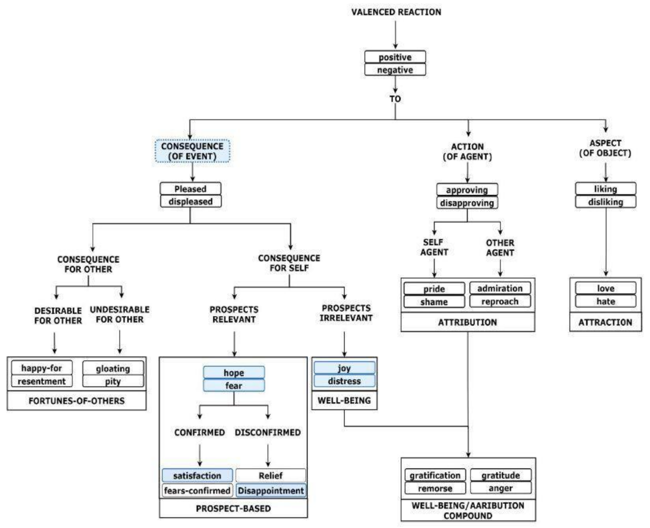


Fig. 1: OCC structure of emotions [17].

Table 1 summarizes OCC specifications of the highlighted emotion types; e.g. the OCC theory defines joy as *is being pleased about a desirable consequence of event*. For example, consider a maze game in which an agent is looking for gold. When the agent finds a room with a gold pile, and it takes one step toward the gold, this has a desirable consequence (the agent is certain that it gets closer to the gold), so the agent feels pleased and as a result it starts to feel joy for the gold. However, satisfaction is different. It is defined as *being pleased about the confirmation of the prospect of a desirable consequence*. This emotion needs achievement confirmation whereas joy can be triggered whenever the agent becomes certain that the goal is achievable, although not fulfilled yet. In the above example, satisfaction is triggered when the agent actually

3

acquires the gold. Additionally, while joy affects satisfaction, the agent might not be satisfied towards every goal which it is joyful about. In the earlier set-up, the agent, when proceeding to collect the gold, faces guardians that need to be defeated first, and ends up consuming a unique item to win the combat. Thus, despite reaching the goal that it is joyful about, it would not be satisfied for failing to keep all its prized possessions.

Table 1: Selected Emotions specifications according to the OCC theory [17].

Joy:	pleased about a desirable consequence of event
Distress:	displeased about an undesirable consequence of event
Hope:	pleased about the prospect of a desirable consequence of event
Fear:	displeased about the prospect of an undesirable consequence of event
Satisfaction:	pleased about the confirmation of the prospect of a desirable consequence
Disappointment:	displeased about the disconfirmation of the prospect of a desirable consequence

In general, dealing with emotions involves *appraisal* and *coping* [17]. When an agent receives an event, the appraisal process is triggered to form emotions. Afterward, the agent responds to those emotions based on coping strategies which affects the agent behavior towards the environment. In other words, emotions regulate the agent's actions during the coping process. In this paper, we focus on modeling of appraisal —the proposed appraisal model of event-based emotions will be presented in Section 4.

3 Agent-based Player Experience Testing Framework

In this section, we will explain the proposed framework architecture with their components and demonstrate appraisal in PX testing with some examples.

3.1 The framework architecture

The general architecture of the proposed framework is presented in Figure 2, showing appraisal model of emotions, player characterization, Aplib and PX evaluation as the key components. They are defined below.

Appraisal model of emotions. Emotions of a test agent are modeled based on the OCC theory. Since the framework does emotional evaluations on perceived events produced by the agent's actions or uncontrollable dynamic events such as hazards, only event-based emotions relevant to computer games are needed. To model these emotions, a *transition system approach* is proposed, which is formalized in Section 4. This calculates the event-based emotion types with their respective intensity. We will focus on a single test agent setup, thus for now we leave out emotions that are only valid in multi-agent settings. There is also room for extending the model, in the future, to test aspects of players' experience that are formed in various social contexts.

Player characterization. Some properties of the appraisal model of emotions need to be specified by game designers with respect to the game under test as well as the player characteristics. For example, the designers should specify what goals are relevant for players (e.g. winning the game, collecting in-game money), what in-game events are relevant to these goals, and in what way they are related to the goals (are they desired

4

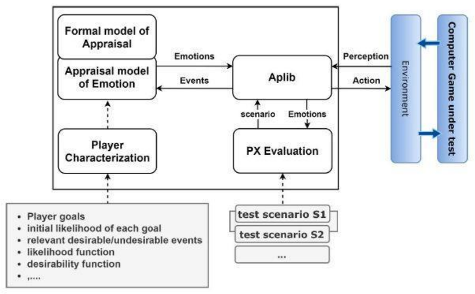


Fig. 2: Automated PX testing framework architecture.

towards reaching a goal, or else undesirable?). Additionally, the desirability of an event might differ from one player character to another. Thus, player or set-up dependent properties must be initially set in this part of framework, before running the model of appraisal. Having such a component in our framework also provides an opportunity to enhance it in the future with more advanced characteristics such as players' moods and play-style (e.g., exploratory or aggressive [25,29]).

Aplib[4] [20]. A Java library for programming intelligent agents. It provides an embedded Domain Specific Language (DSL) to use all benefits of the Java programming language. Aplib has a BDI architecture [12] with a novel layer for tactical programming to control agents behavior more abstractly. Despite other use cases, the library has been developed for testing tasks in highly interactive software like games.

PX Evaluation. Designers give test scenarios to the framework to check whether their newly developed content indeed triggers the expected emotions. This part is responsible for the visualization of the emotional state of the test agent as it pursues dedicated goals in a game environment with a given test scenario. Generated emotion types with their upward/downward trends during the test would assist designers to alter game parameters to optimize the experience in a certain degree.

3.2 Appraisal theory in PX testing

As mentioned earlier, the appraisal process is an essential part of computational models of emotions. So, to automatically test the player experience based on emotions, we need to include this process in our framework for creating emotions. This would allow us to check whether the designers' expected emotions are as same as the triggered players' emotions when exposed to certain situations in the game.

For instance, educational games are often evaluated based on the engagement level of learners to promote learning. Traditionally, to do this, players' emotions are tracked using either self-reports or automated facial emotion detection during a game-based task [16], Identifying positive and negative emotions plays an essential role in deciding

[4] https://iv4xr-project.github.io/aplib/

5

if some game-based conditions and tasks need to be changed to optimize learning. Our proposed framework would help in performing this process automatically using model of emotions to create emotions with respect to events.

Users of a more traditional, non-game, system typically need to feel higher levels of positive emotions and low levels of negative emotions to reach a satisfactory experience, while moderate levels of positive emotions and a high level of negative emotions such as distress, fear and disappointment could end up in an unsatisfactory experience with the system [18]. These negative emotions reflect users' feelings when they are unable or unsure of how to use the system in some situations. This lead to the poor usability of the system [23]. However, computer games, e.g. those in the RPG and combat genres, can be deliberately designed to invoke certain negative emotions for certain experience in players because it can ultimately contribute to their enjoyment [5] or even lead to high level of positive emotion when the player overcomes reasons that evoked negative emotions like fear and frustration [14]. Thus, unlike UX testing, in PX testing designers also need be able to analyze relations between positive and negative emotions. Our proposed framework can automatically check whether these emotions are appraised during playing the game. The prototype further refines this by also tracking when and where these emotions occur, thus enabling refined analyses. If the patterns of these emotions do not meet expectation, designers can change properties of the game and iterate the emotional testing process to achieve the expected emotions.

Ultimately, modelling a player's coping process improve the ability of the framework in PX testing. This is discussed briefly in Section 7. However, being able to model the coping behavior does not change the fact that the framework needs to also support the appraisal process of emotions in the first place. For this reason, our proposed framework first focuses on the appraisal process.

4 Event-based Formal Model of Emotion

Imagine that a software testing agent which takes the role of the player is deployed on a computer game to do PX testing. The agent is modelled as an event-based transition system which can appraise emotions to emulate the emotional state of a player. Its state consists of its 'belief' (perception) over the game and its emotions which can eventually affect its behavior to resemble the player behavior. In this section, we describe the essential part of the formalization of this event-based emotion transition system to conduct an approach for formal modeling of automated PX testing.

In the following, we assume an agent to have beliefs and goals, based on which it decides which actions should be taken in the environment. Being able to differentiate between different goals is useful for PX testing, as games often offer various optional plots and goals to players to improve their non-linearity and replay value. A goal g is represented as a pair $\langle id, x\rangle$, with id as its unique identifier and x as its significance or priority of the goal. Goals and their significance are static in this setup. We also assume that an agent senses its environment by means of events. For simplicity, it is assumed that the agent observes one event at a time, causing the agent to transition from one state to another. Whereas the agent's own actions are events, there are also events that arise from environmental dynamism such as hazards and updates by dynamic objects.

6

We also add the event *tick* to discretely represent the passing of time. We represent emotion types as $Etype = \{\ Joy, Distress, Hope, Fear, \ldots\}$. In the sequel, *etype* ranges over this set.

Definition 1 An emotional testing agent is represented by a transition system M, described by a tuple:

$$\langle \Sigma, s_0, G, E, \delta, \Pi, Thres \rangle$$

where:

– G is a set of the agent's goals.
– Σ is the set of M's possible states. Each state s in Σ is a pair $\langle K, Emo \rangle$ where:
  - K is a set of propositions representing the agent's beliefs. We additionally require that for every $g \in G$, K includes a proposition representing the goal's confirmation or dis-confirmation status, and a proposition representing the likelihood of reaching this goal from the current state. The former is represented by $status(g, p)$ where $p \in \{achieved, failed, proceeding\}$ and the latter by $likelihood(g, v)$ where $v \in [0..1]$.
  - Emo is a set containing the agent's active emotions, each is represented by a tuple $\langle etype, w, g, t_0 \rangle$ specifying the emotion type *etype*, its intensity w with respect to a goal g, and the time t_0 at when the emotion is triggered.
– $s_0 \in \Sigma$ is the initial state. It should specify the agent's initial belief on the likelihood of every goal, as well as initial prospect-based emotions (hope and fear). The rationale for the latter is that having an initial prospect towards a goal implies that there is also hope for achieving it, as well as some fear of its failure.
– E is the set of events the agent experiences.
– $\delta : \Sigma \times E \rightarrow \Sigma$ is the state transition function that describes how M moves from one state to another upon perceiving an event. The definition is rather elaborate, and will be given separately in Definition 2.
– $\Pi = \langle Des, Praisew, DesOther, Liking \rangle$ is a tuple of appraisal dimensions according to the OCC theory. This determines how an event is appraised in terms of its *desirability*, *praiseworthiness*, *desirability by others* and *liking*.
– *Thres* is a set of thresholds, one for every type of emotion.

As an example, Figure 3 illustrates first few transitions. We, additionally, assume the agent maintains an emotional memory, called *emhistory*, which keeps the history of active emotions (*Emo*) for a reasonable time window in the past:

$$emhistory \;=\; \overbrace{Emo_{t-d}\ ,\ \ldots\ ,\ Emo_{t-1}}^{\text{time window } d}$$

where t is the current system time and d is the size of the memory's time window. Emo_{t-i} indicates the active emotional at time $t - i$ in the past.

Before presenting the rest of the formal model, we feel the necessity to bring more clarity into the concept of goals' likelihood and status. The transition system is defined in a way that there is a slight difference between $likelihood(g, 1)$ and $status(g, achieved)$. When an agent experiences $likelihood(g, 1)$, it is possible that the

7

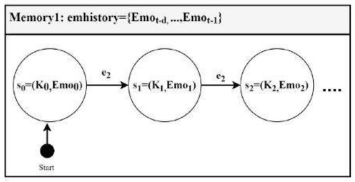


Fig. 3: An agent's state transitions, as it receives an event e_1 followed by e_2.

goal g does not get confirmed in the same state. In other words, the agent comes to believe that the goal is reachable with 100% certainty, but the achievement of the goal has not been confirmed yet in the current state. A similar relation holds for $likelihood(g,0)$ and $status(g,failed)$.

The next key point is the agent's appraisal component Π, which has four dimensions. They help in modeling how events are appraised with respect to every goal in the corresponding dimension. Each appraisal dimension is described as a function over the agent's beliefs, an event and a goal: $\Pi_{\mathbf{Dim}}(K,e,g)$, where $Dim \in \{Des, Praisew, DesOther, Liking\}$. For example, $\Pi_{\mathbf{Des}}(K,e,g)$ determines the desirability of an event e with respect to the goal g, judged when the agent believes K; the latter implies that this desirability might change when K changes. Depending on the emotion, one or multiple appraisal dimensions might be triggered. Currently, $\Pi_{\mathbf{Des}}$ is the only dimension being actively used in our model because according to the OCC theory, the only appraisal dimension which affects our selected emotion types is the desirability function. However, we keep the structure in the general form for possible future extension of the emotion types.

Below we will explain how emotions will be calculated, but importantly we should note that PX designers must provide some information as well, namely the following components of the tuple in Definition 1: (1) the goal set G, along with the significance and initial likelihood of each goal ($likelihood(g, v_{init})$), (2) likelihood functions modelling how events affect the agent's belief towards goals' likelihood, (3) the appraisal dimensions, in particular $\Pi_{\mathbf{Des}}(K,e,g)$, (4) the thresholds $Thres$ and (5) decay rate $decay_{etype}$. In the simplest form, $\Pi_{\mathbf{Des}}(K,e,g)$ can be described by a mapping that maps events to the goals they are perceived as desirable/undesirable. In a more refined description this can be a function that monotonically increases with respect to the goal significance and likelihood. In terms of the architecture in Figure 2, the above components are described in the *Player Characterization* part.

Definition 2 *Event-based Transition.* As mentioned earlier, the agent's state transition is driven by one incoming event at a time. The transition function (δ in Def. 1) is defined as follows. Let e be an occurring event:

$$\langle K\ ,\ Emo\rangle \xrightarrow{e} \langle K'\ ,\ \overbrace{newEmo(K,e,G)\ \oplus\ decayedEmo(Emo)}^{\text{updated emotion } Emo'}\rangle$$

where:

8

- $K' = e(K) \setminus H$, where $e(K)$ is the agent's new beliefs obtained by updating K with event e; here, the event e is assumed to have a semantic interpretation as a function that affects K, including the parts that concern goals' likelihood and status. H expresses likelihood information that can be removed from $e(K)$, because the corresponding goals are achieved or failed. More precisely, H is the set $\{\ likelihood(g,v) \mid status(g,p) \in e(K),\ p \in \{achieved, failed\},\ v \in \{0,1\}\ \}$.
- $Emo' = newEmo(K,e,G) \oplus decayedEmo(Emo)$ is the agent's emotions updated by the perceived event e and the agent's new beliefs. Importantly, the $newEmo(K,e,G)$ specifies the *newly* triggered emotions (see Def.3), whereas $decayedEmo(Emo)$ (see Def.4.1) is a set of active emotions that decay over time. The operator $\oplus$ merges all these emotions after applying some constraints to have the updated emotional state of the agent. The emotional update is explained in Section 4.3.

When an agent perceives an event (except *tick* event), new emotions may be triggered. This is done by calculating a so-called 'emotion function' $\mathcal{E}$ for every emotion type, as follows:

$$\mathcal{E}_{etype}(K,e,g) = w$$

This function specifies the activation intensity w of the emotion $etype$ towards the goal g, as a consequence of the occurrence of e and having beliefs K. Importantly, note that the function expresses *goal oriented* emotions, whereas the OCC theory includes e.g. emotions towards events or objects. We focus on goal oriented emotions due to the importance of goals, ranging from defeating monsters to getting the highest score, for game players. A *tick* event is used to represent the passing of time. This event would cause decays of active emotions in the transition system. The definition of newly triggered emotion, mentioned in Def.2, is given below. It is used whenever a new emotion is triggered or an existing emotion reoccurs in the system. The way these new emotions are merged with existing emotions in Emo, as mentioned in Def.2, will be explained in Section 4.3. We also need to remind that some hope and fear already exist in the system at the beginning which can be re-triggered by this function. Their initial values are set according to goals' significance and initial likelihoods of goals.

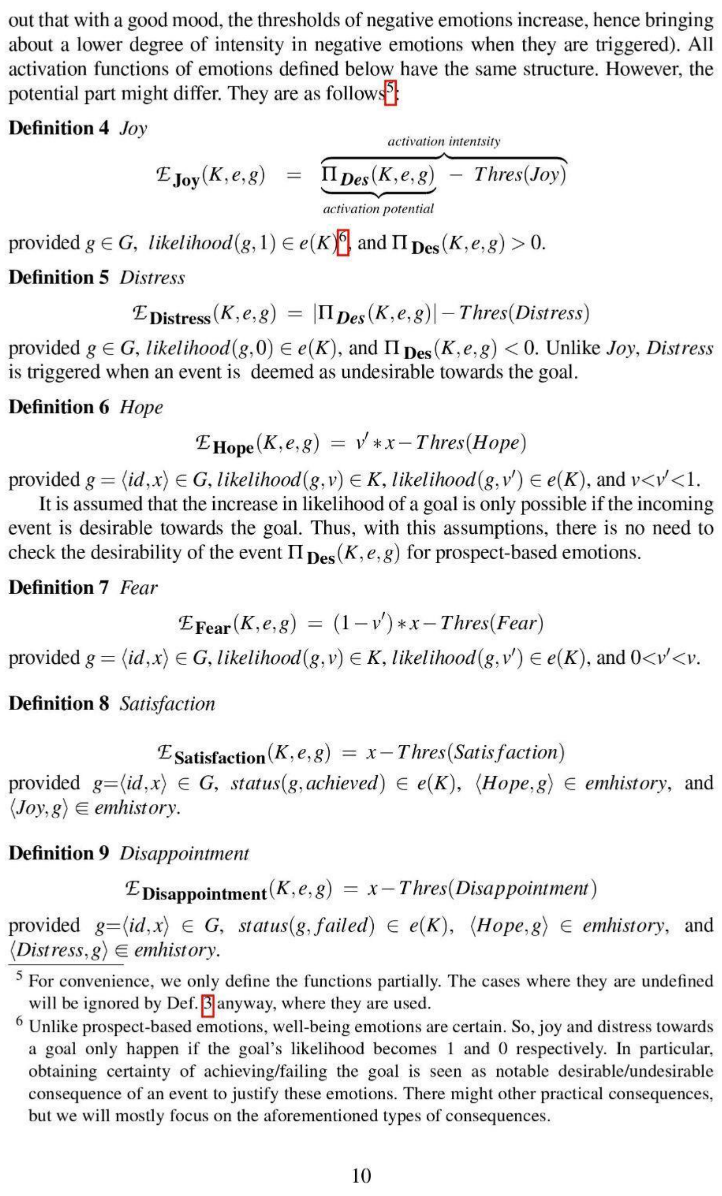
Definition 3 *New Emotions.* The set of new emotions triggered by e is:

$$newEmo(K,e,G) = \{\langle etype, g, w, t\rangle \mid etype \in Etype,\ g \in G,\ w = \mathcal{E}_{etype}(K,e,g) > 0\}$$

where t is the current system time that the emotion is triggered.

In the above definiton E_{etype} is a so-called activation emotion function that calculates the activation intensity for different newly triggered event-based emotion types. Each activation emotion function has an activation potential and a threshold which form the activation intensity of the newly triggered emotion (see Def.4). The level of desirability an event respecting a goal and the agent's goal likelihood are the main variables affecting the activation potential as hinted in the OCC theory. To trigger a new emotion type, its activation potential value needs to pass the corresponding threshold. The concept of threshold is needed if we want to support setups with different agent's moods because the thresholds depend on the moods (e.g. Steunebrink et al.[26] pointed

9



out that with a good mood, the thresholds of negative emotions increase, hence bringing about a lower degree of intensity in negative emotions when they are triggered). All activation functions of emotions defined below have the same structure. However, the potential part might differ. They are as follows[5]:

Definition 4 *Joy*

$$\mathcal{E}_{\mathbf{Joy}}(K,e,g) \;=\; \overbrace{\underbrace{\Pi_{\boldsymbol{Des}}(K,e,g)}_{\textit{activation potential}} \;-\; Thres(Joy)}^{\textit{activation intentsity}}$$

provided $g \in G$, $likelihood(g,1) \in e(K)$[6], and $\Pi_{\mathbf{Des}}(K,e,g) > 0$.

Definition 5 *Distress*

$$\mathcal{E}_{\mathbf{Distress}}(K,e,g) \;=\; |\Pi_{\boldsymbol{Des}}(K,e,g)| - Thres(Distress)$$

provided $g \in G$, $likelihood(g,0) \in e(K)$, and $\Pi_{\mathbf{Des}}(K,e,g) < 0$. Unlike *Joy*, *Distress* is triggered when an event is deemed as undesirable towards the goal.

Definition 6 *Hope*

$$\mathcal{E}_{\mathbf{Hope}}(K,e,g) \;=\; v' * x - Thres(Hope)$$

provided $g = \langle id,x\rangle \in G$, $likelihood(g,v) \in K$, $likelihood(g,v') \in e(K)$, and $v{<}v'{<}1$.

It is assumed that the increase in likelihood of a goal is only possible if the incoming event is desirable towards the goal. Thus, with this assumptions, there is no need to check the desirability of the event $\Pi_{\mathbf{Des}}(K,e,g)$ for prospect-based emotions.

Definition 7 *Fear*

$$\mathcal{E}_{\mathbf{Fear}}(K,e,g) \;=\; (1-v') * x - Thres(Fear)$$

provided $g = \langle id,x\rangle \in G$, $likelihood(g,v) \in K$, $likelihood(g,v') \in e(K)$, and $0{<}v'{<}v$.

Definition 8 *Satisfaction*

$$\mathcal{E}_{\mathbf{Satisfaction}}(K,e,g) \;=\; x - Thres(Satisfaction)$$

provided $g{=}\langle id,x\rangle \in G$, $status(g,achieved) \in e(K)$, $\langle Hope,g\rangle \in emhistory$, and $\langle Joy,g\rangle \in emhistory$.

Definition 9 *Disappointment*

$$\mathcal{E}_{\mathbf{Disappointment}}(K,e,g) \;=\; x - Thres(Disappointment)$$

provided $g{=}\langle id,x\rangle \in G$, $status(g,failed) \in e(K)$, $\langle Hope,g\rangle \in emhistory$, and $\langle Distress,g\rangle \in emhistory$.

[5] For convenience, we only define the functions partially. The cases where they are undefined will be ignored by Def. 3 anyway, where they are used.

[6] Unlike prospect-based emotions, well-being emotions are certain. So, joy and distress towards a goal only happen if the goal's likelihood becomes 1 and 0 respectively. In particular, obtaining certainty of achieving/failing the goal is seen as notable desirable/undesirable consequence of an event to justify these emotions. There might other practical consequences, but we will mostly focus on the aforementioned types of consequences.

10

4.1 Decay of emotions

Every emotion has a duration called *emotion episode* in which the peak of its intensity, its decay rate, possible recurrences, and the time that the emotion is triggered are shown[26]. As indicated earlier in Def.1, *tick* is a time event to show the passing of time in our transition system. We can reflect decays of emotions using this event:

$$\langle K, Emo\rangle \xrightarrow{e=tick} \langle K', Emo'\rangle$$

where K' and Emo' refer to the updated beliefs and updated active emotions after the transition. The intensity of active emotions in Emo would decrease as follows:

$$\begin{aligned} &decayedEmo(Emo) \ = \\ &\quad \{ \ \langle etype, g, w', t_0\rangle \ | \ \ \langle etype, g, w, t_0\rangle \in Emo, \ \ w' = \mathsf{intensitydecay}_{\mathbf{etype}}(w_0, t_0) > 0, \\ &\qquad\qquad\qquad\qquad\quad w_0 = emhistory(etype, g, t_0) \ \} \end{aligned}$$

where $w_0 = emhistory(etype, g, t_0)$ denotes the initial intensity of $etype$ with respect to g which can be obtained from $emhistory$. There is not a unique quantitative formalization for the decay function $\mathsf{intensitydecay}$. This function can be defined in a way which relates the usage and the interpretation of decay [27] [6]. However the peak of intensity (w_0), the time at which the emotion is triggered (t_0) and the decay rate ($decay_{etype}$) are essential parameters that must be taken into account. While an inverse sigmoid decay function is proposed by [27] to reflect the gradual decrease of intensities, [6] is making use of a negative exponential function with almost the ame parameters. We used the latter decay function [6] in our model although the sigmoid decay function [27] can be used as well.

$$\mathsf{intensitydecay}_{\mathbf{etype}}(w_0, t_0) = w_0 \ * \ e^{\,c \,*\, decay_{etype} \,*\, (t-t_0)} \quad , -1 < c < 0$$

where t is the current system time and t_0 is the time at which the emotion starts.

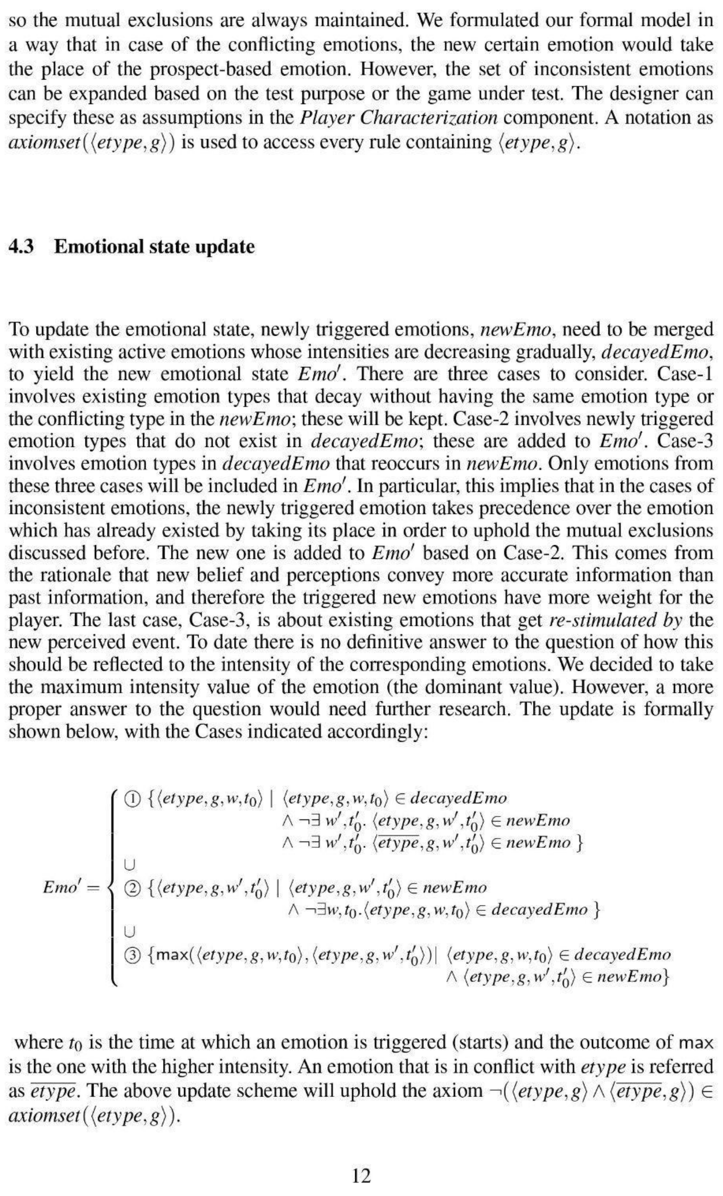
4.2 Inconsistent emotions

Emotions are triggered regarding the goals, so technically the agent might have several emotions towards the same goal. Nevertheless, the OCC theory states that some emotions are mutually exclusive which means a human can not have them simultaneously for the same goal [26]. These mutual exclusions, which should then also be held in every state of our transition system, are as follows:

$$\begin{aligned} Emo' &\models \neg(\langle Hope, g\rangle \wedge \langle Joy, g\rangle) \\ Emo' &\models \neg(\langle Fear, g\rangle \wedge \langle Distress, g\rangle) \end{aligned}$$

As it is explained in Section 2, whereas emotions such as hope and fear are prospect-based emotions which means they are uncertain ($likelihood(g, v)$), emotions like joy and distress are certain [26], so it is illogical to have both in the system. For example, when a player is joyful of acquiring the key to an in-game treasure room, because now the treasure should certainly be within his/her reach, this joy would now replace what was merely hope for getting the treasure. In general, in case of happening a certain emotion, it replaces the corresponding prospect-based emotion,

11



so the mutual exclusions are always maintained. We formulated our formal model in a way that in case of the conflicting emotions, the new certain emotion would take the place of the prospect-based emotion. However, the set of inconsistent emotions can be expanded based on the test purpose or the game under test. The designer can specify these as assumptions in the *Player Characterization* component. A notation as $axiomset(\langle etype, g\rangle)$ is used to access every rule containing $\langle etype, g\rangle$.

4.3 Emotional state update

To update the emotional state, newly triggered emotions, *newEmo*, need to be merged with existing active emotions whose intensities are decreasing gradually, *decayedEmo*, to yield the new emotional state Emo'. There are three cases to consider. Case-1 involves existing emotion types that decay without having the same emotion type or the conflicting type in the *newEmo*; these will be kept. Case-2 involves newly triggered emotion types that do not exist in *decayedEmo*; these are added to Emo'. Case-3 involves emotion types in *decayedEmo* that reoccurs in *newEmo*. Only emotions from these three cases will be included in Emo'. In particular, this implies that in the cases of inconsistent emotions, the newly triggered emotion takes precedence over the emotion which has already existed by taking its place in order to uphold the mutual exclusions discussed before. The new one is added to Emo' based on Case-2. This comes from the rationale that new belief and perceptions convey more accurate information than past information, and therefore the triggered new emotions have more weight for the player. The last case, Case-3, is about existing emotions that get *re-stimulated by* the new perceived event. To date there is no definitive answer to the question of how this should be reflected to the intensity of the corresponding emotions. We decided to take the maximum intensity value of the emotion (the dominant value). However, a more proper answer to the question would need further research. The update is formally shown below, with the Cases indicated accordingly:

$$Emo' = \begin{cases} \text{①}\ \{\langle etype, g, w, t_0\rangle \mid \langle etype, g, w, t_0\rangle \in decayedEmo \\ \qquad\qquad\qquad\qquad \wedge \neg\exists\, w', t_0'.\ \langle etype, g, w', t_0'\rangle \in newEmo \\ \qquad\qquad\qquad\qquad \wedge \neg\exists\, w', t_0'.\ \langle \overline{etype}, g, w', t_0'\rangle \in newEmo\ \} \\ \cup \\ \text{②}\ \{\langle etype, g, w', t_0'\rangle \mid \langle etype, g, w', t_0'\rangle \in newEmo \\ \qquad\qquad\qquad\qquad \wedge \neg\exists w, t_0.\langle etype, g, w, t_0\rangle \in decayedEmo\ \} \\ \cup \\ \text{③}\ \{\mathsf{max}(\langle etype, g, w, t_0\rangle, \langle etype, g, w', t_0'\rangle) \mid \langle etype, g, w, t_0\rangle \in decayedEmo \\ \qquad\qquad\qquad\qquad\qquad\qquad \wedge \langle etype, g, w', t_0'\rangle \in newEmo\} \end{cases}$$

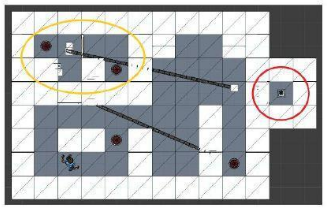
where t_0 is the time at which an emotion is triggered (starts) and the outcome of max is the one with the higher intensity. An emotion that is in conflict with *etype* is referred as $\overline{etype}$. The above update scheme will uphold the axiom $\neg(\langle etype, g\rangle \wedge \langle \overline{etype}, g\rangle) \in axiomset(\langle etype, g\rangle)$.

12

5 Proof of Concept

We conducted our experiment on a game called Lab Recruits[7] which we subject to the combination of aplib and our implemented model of appraisal[8] to provide the proof of concept and show our early results in PX testing. Lab Recruits is a 3D game developed in Unity which has different replayable levels. Each level is a laboratory building with a number of rooms containing interactable objects, such as button and non-interactable objects, such as desk and fire hazards.

Figure 4a shows the floor plan of the level exposed to PX testing using our approach. It consists of four buttons, three doors, and some fire hazards. The goal is for the player to escape the level by reaching the exit room circled in red. Access to this room is guarded by a closed 'final door'. The level contains some rooms with a puzzle (yellow circle) that involves finding the buttons to open the final door and reopen the doors that in the process become closed to entrap the agent. Figure 4b and 4c show two provided setups with the different amount and locations of fire hazards. The agent will lose health points by passing each fire hazard. These setups are examples of choices considered by designers, although being currently simple, as to which one would lead to better PX.


(a) The floor plan of the level.

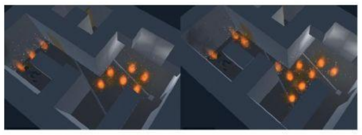

(b) Setup 1. (c) Setup 2.

Fig. 4: The level under the PX test in Lab Recruits.

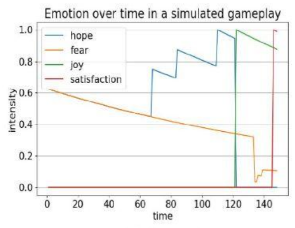
As mentioned in Section 4, a developer sets needed inputs of the model such as the goal set, initial likelihood of each goal, the desirability of events for each goal, the threshold and decay rate of emotions in *Player Characterization*. A test agent is deployed, set with multiple goals, though here we will only discuss the most significant one, namely completing the level. Initially, the agent is assumed to believe that the likelihood of achieving this goal is 0.5. The agent is given a program so that it can automatically explore the level. As the agent progresses, its belief on the likelihood of completing the level changes, depending on the number of opened door as well as remained closed doors. Opening each door is assumed to have a desirable consequence for the agent because it increases the chance of the agent to complete the level.

The timeline of triggered emotions in the agent with respect to the goal "completing the level" is shown in Figure 5, along with their intensity levels at each time. The agent

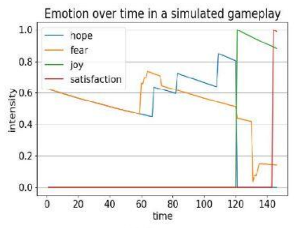
[7] https://github.com/iv4xr-project/iv4xrDemo/tree/occDemoPrototype
[8] https://github.com/iv4xr-project/jocc

13

initially experience some hope and fear due to the assumed initial belief that completing the game is possible, with the likelihood 0.5. In both setups, when the agent pushes the button that opens the first door (time=70[9]), the agent's hope regarding completing the game starts to increase. It decays or gets re-stimulated according to the events until time 120 when it is replaced by joy. The agent feels a level of satisfaction, When completing the game. Comparing two setups reveals something interesting. Fear shows a quite different trend in setup 2 (Figure 5b). It is strongly stimulated multiple times during the execution, whereas the same emotion is rarely stimulated in setup 1, and when it happens, it happens with much less intensity (Figure 5a), towards the end of the play, where it matters less. Such comparison can be useful for designers e.g. to determine the amount, and placement, of hazards to induce certain degree of fear along with keeping the chance for satisfactory experience of accomplishing the goal. In our case, setup 1 is less likely to thrill the player, whereas setup 2 has a better balance of the quantity and placement of the fire, by generating fear, while still keeping the level survivable.



(a) Setup 1



(b) Setup 2

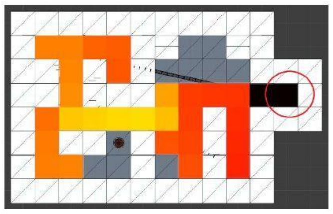
Fig. 5: The emotions' timelines correspond two setups of the game level in Figure. 4. (threshold=0, decay rate=0.005)

Figure 6 shows some heat maps, providing spatial information of the agent's emotions in Setup 2. Comparing the outcomes of Figures 5b and 6a illustrates that the highest level of fear is experienced between time 65 to 75 when the agent is in a particular fire covered corridor (yellow in Figure 6a). Fire intensifies the agent's fear of failure, and moreover the agent has to walk this corridor several times. The most drastic decline in fear is when the agent is about to finish the level.

As can be seen in Figure 6b, the agent feels a higher level of hope when progressing in solving the buttons-doors puzzle in the puzzle rooms. After pushing the button that corresponds to the final door and reopening the door of puzzle room to escape it, the agent becomes certain that passing the final door is achievable now. Thus, the hope

[9] The system is event driven, so only events can change the likelihoods. All emotions decay until an event is perceived. However, we can add an event type to the system to decay the likelihoods when there is no event for some period of time to update the emotional state.

14


(a) Negative emotions: yellow= high fear, Orange shades=low fear.

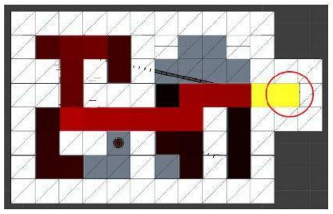

(b) Positive emotions: Mahogany red=hope, Ruby red= joy, yellow= satisfaction.

Fig. 6: The heat maps of triggered emotions in setup 2.
Black=no emotion, white= walls (not walkable), gray=unexplored area.

suddenly is replaced by the joy for reaching the final door to complete the game. At the end, the agent feels satisfied when the achievement is confirmed. Having such information would help Lab Recruits designers to adjust the puzzles and fire hazards in such a way to induce certain emotions, at the right moments and the right places, which ultimately affect a certain aspect of player experience like enjoyment.

6 Related Work

PX researchers aim to understand the gaming experience to ultimately induce certain experience. Fernandez [9] outlines the influence of players' emotional reactions and their profile in enjoyment by extending the usability methods to uncover relationships between game components and the degree of fun in players. Sanchez et.al [24] explained that usability of games can be defined in the term of *playablity*. They present a framework guided by attributes and properties of playability to characterise experience for PX evaluation and observing the relation between the experience and the developed elements of a commercial video game. Psycho-physiological methods is among techniques to measure aspects of PX like flow and immersion. Jennett's et al. [13] tries to develop a subjective and objective measure for immersion using questionnaires and eye movement tracking respectively. Drachen et al. [7] report a significant the correlation between heart rate, electrodermal activity and the self-reported experience of players in first-person shooter games. Zook and Riedl [30] introduce a temporal data-driven model to to predict the impact of game difficulty to player experience. Results of their empirical study on a role-player combat game show the game, that tailors its difficulty to fit a player abilities, improves the player experience. Most of PX prediction techniques are data-driven which involve human players in the process and as a result, they demand a high level of human labor. This led researchers to investigate model-driven approaches. A computational model of motivation is presented in [11] to predict PX without the need of human player using empowerment, the degree of control an agent has over the game. The study measures empowerment by intelligent agents to

15

create levels with defined empowerment to induce different PX. This would help to produce desired content characteristics during the procedural content generation.

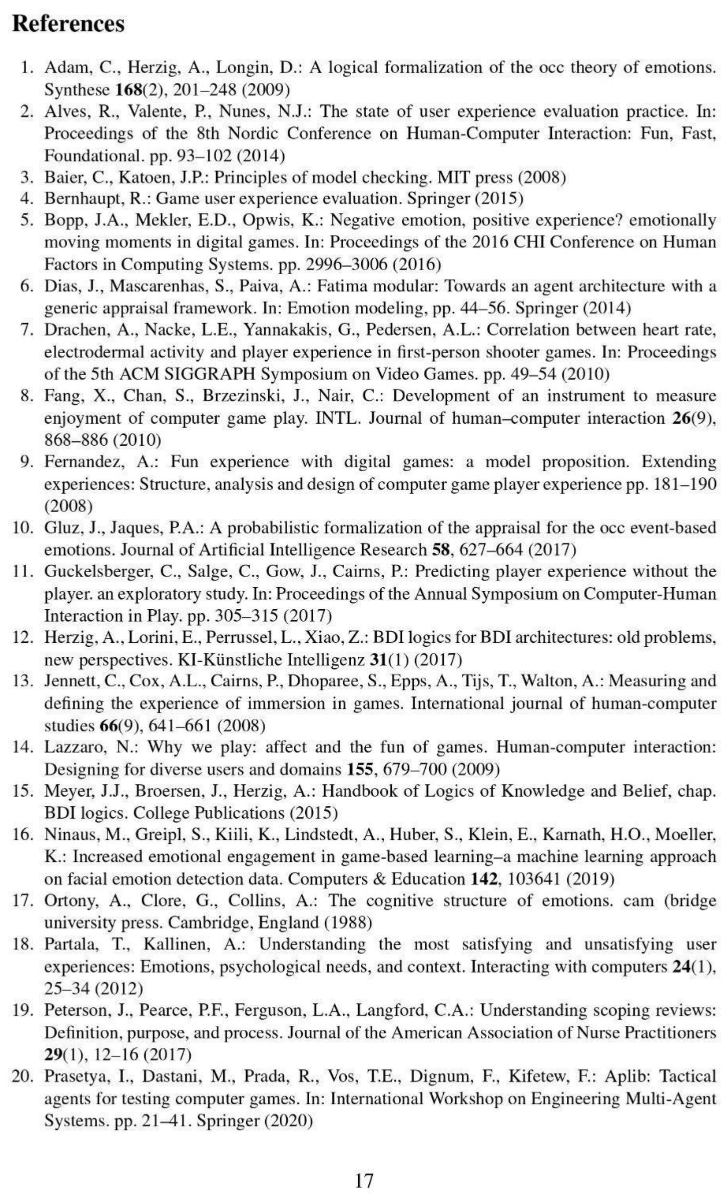
Despite existing research on modeling the OCC theory, the theory has not been employed in the context of PX testing. Having a proper formalization of emotion would act as a bridge from psychological description of emotions to computational models of emotions which are translatable to codes. Formalization of emotions has been mostly done in the form of BDI logic. Steunebrink [26] deployed a formal model inspired by the OCC theory to specify the influence of emotions, specifically hope and fear, on a BDI agent's decisions. Later, a full version of the model with all 22 emotions is explained in [27]. Dias et al.[6] presents an OCC-based appraisal engine called FAtiMA (**F**earnot **A**ffec**TI**ve **M**ind **A**rchitecture) for creating autonomous agent characters that can appraise events and behave based on socio-emotional skills. Its main use case is to automate virtual characters in conversing with humans. FAtiMA is claimed to be inspired by the OCC theory to simulate emotional skills in autonomous agents. However, so far, no formal model has been introduced to evaluate the toolkit regarding the OCC theory. A BDI-like probabilistic formalization is described in [10] for OCC event-based emotions during the appraisal. The study evaluates the desirability of consequences of an event based on the agent's goal and the degree that the consequence can improve the possibility of the goal achievement. Unlike other formalisations that give a high level function for appraisal variables, it proposes a more refined logic-base calculation for these variables and also tries to formalize 'effort' and 'realization' that are involved in appraising some event-base emotions.

7 Conclusion & Future Work

This paper presented an automated PX testing approach using an emotional model. An event-based transition system is introduced to model the appraisal for event-based emotions according to the OCC theory which is then combined to a Java library for tactical agent programming called aplib to create an agent-based PX testing framework. Early results of our experiment with the prototype show that such a framework that can emulate players' emotions would let developers to investigate how emotions of players would evolve in the game during the development stage. By providing e.g. heat-map visualisations of triggered emotions and their timelines, designers gain insight on how to alter parameters of their systems to evoke certain emotions.

We are currently doing more advanced experiments using the case study, Lab Recruits, to investigate initial moods, emotions and their effect on certain aspects of PX as a future work. There are also some concepts like emotional intensity after a recurrence that are described with high level functions in the literature which need a calculation mechanism. In particular, we want to do further research on how exactly an emotion should regain its intensity level after a re-stimulation. Furthermore, the proposed framework, if enhanced by the coping process, would be able to simulate the effect of emotions on players' behavior for further PX evaluations. However, this needs extension in our event-based transition system to support the coping process formally respecting the OCC theory. We ultimately plan to conduct research on validation of our model by comparing our results with the data of human players.

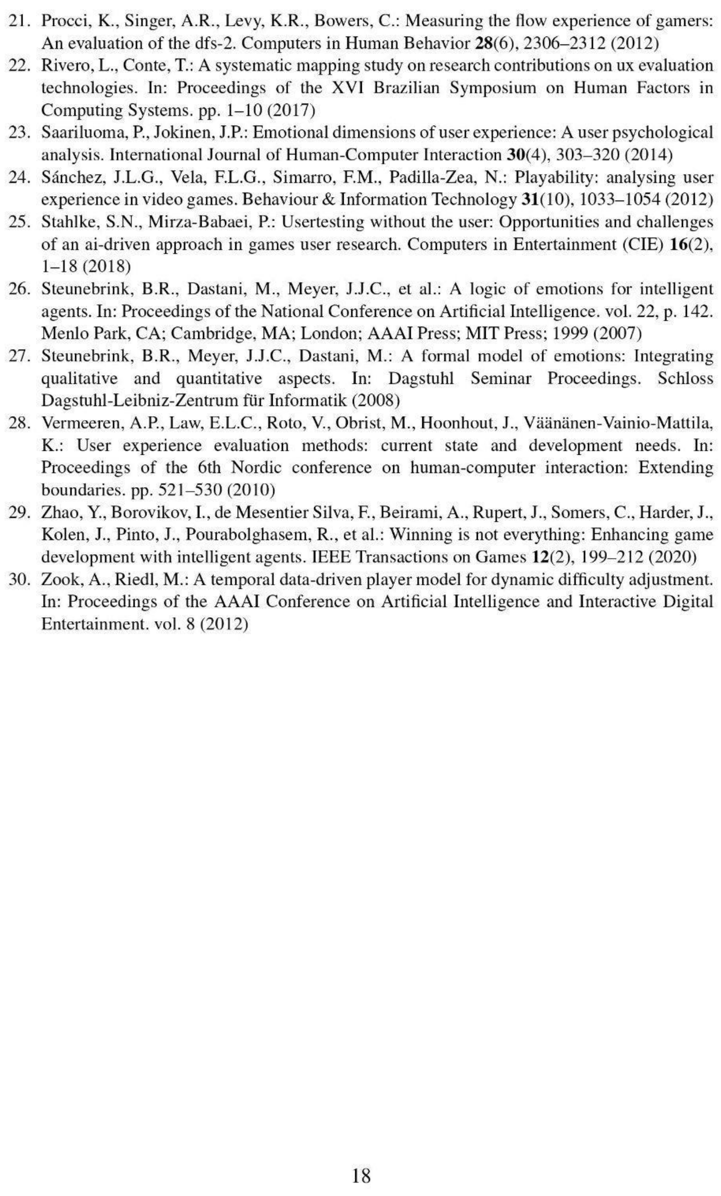
16



References

1. Adam, C., Herzig, A., Longin, D.: A logical formalization of the occ theory of emotions. Synthese **168**(2), 201–248 (2009)
2. Alves, R., Valente, P., Nunes, N.J.: The state of user experience evaluation practice. In: Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational. pp. 93–102 (2014)
3. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
4. Bernhaupt, R.: Game user experience evaluation. Springer (2015)
5. Bopp, J.A., Mekler, E.D., Opwis, K.: Negative emotion, positive experience? emotionally moving moments in digital games. In: Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems. pp. 2996–3006 (2016)
6. Dias, J., Mascarenhas, S., Paiva, A.: Fatima modular: Towards an agent architecture with a generic appraisal framework. In: Emotion modeling, pp. 44–56. Springer (2014)
7. Drachen, A., Nacke, L.E., Yannakakis, G., Pedersen, A.L.: Correlation between heart rate, electrodermal activity and player experience in first-person shooter games. In: Proceedings of the 5th ACM SIGGRAPH Symposium on Video Games. pp. 49–54 (2010)
8. Fang, X., Chan, S., Brzezinski, J., Nair, C.: Development of an instrument to measure enjoyment of computer game play. INTL. Journal of human–computer interaction **26**(9), 868–886 (2010)
9. Fernandez, A.: Fun experience with digital games: a model proposition. Extending experiences: Structure, analysis and design of computer game player experience pp. 181–190 (2008)
10. Gluz, J., Jaques, P.A.: A probabilistic formalization of the appraisal for the occ event-based emotions. Journal of Artificial Intelligence Research **58**, 627–664 (2017)
11. Guckelsberger, C., Salge, C., Gow, J., Cairns, P.: Predicting player experience without the player. an exploratory study. In: Proceedings of the Annual Symposium on Computer-Human Interaction in Play. pp. 305–315 (2017)
12. Herzig, A., Lorini, E., Perrussel, L., Xiao, Z.: BDI logics for BDI architectures: old problems, new perspectives. KI-Künstliche Intelligenz **31**(1) (2017)
13. Jennett, C., Cox, A.L., Cairns, P., Dhoparee, S., Epps, A., Tijs, T., Walton, A.: Measuring and defining the experience of immersion in games. International journal of human-computer studies **66**(9), 641–661 (2008)
14. Lazzaro, N.: Why we play: affect and the fun of games. Human-computer interaction: Designing for diverse users and domains **155**, 679–700 (2009)
15. Meyer, J.J., Broersen, J., Herzig, A.: Handbook of Logics of Knowledge and Belief, chap. BDI logics. College Publications (2015)
16. Ninaus, M., Greipl, S., Kiili, K., Lindstedt, A., Huber, S., Klein, E., Karnath, H.O., Moeller, K.: Increased emotional engagement in game-based learning–a machine learning approach on facial emotion detection data. Computers & Education **142**, 103641 (2019)
17. Ortony, A., Clore, G., Collins, A.: The cognitive structure of emotions. cam (bridge university press. Cambridge, England (1988)
18. Partala, T., Kallinen, A.: Understanding the most satisfying and unsatisfying user experiences: Emotions, psychological needs, and context. Interacting with computers **24**(1), 25–34 (2012)
19. Peterson, J., Pearce, P.F., Ferguson, L.A., Langford, C.A.: Understanding scoping reviews: Definition, purpose, and process. Journal of the American Association of Nurse Practitioners **29**(1), 12–16 (2017)
20. Prasetya, I., Dastani, M., Prada, R., Vos, T.E., Dignum, F., Kifetew, F.: Aplib: Tactical agents for testing computer games. In: International Workshop on Engineering Multi-Agent Systems. pp. 21–41. Springer (2020)

17



21. Procci, K., Singer, A.R., Levy, K.R., Bowers, C.: Measuring the flow experience of gamers: An evaluation of the dfs-2. Computers in Human Behavior **28**(6), 2306–2312 (2012)
22. Rivero, L., Conte, T.: A systematic mapping study on research contributions on ux evaluation technologies. In: Proceedings of the XVI Brazilian Symposium on Human Factors in Computing Systems. pp. 1–10 (2017)
23. Saariluoma, P., Jokinen, J.P.: Emotional dimensions of user experience: A user psychological analysis. International Journal of Human-Computer Interaction **30**(4), 303–320 (2014)
24. Sánchez, J.L.G., Vela, F.L.G., Simarro, F.M., Padilla-Zea, N.: Playability: analysing user experience in video games. Behaviour & Information Technology **31**(10), 1033–1054 (2012)
25. Stahlke, S.N., Mirza-Babaei, P.: Usertesting without the user: Opportunities and challenges of an ai-driven approach in games user research. Computers in Entertainment (CIE) **16**(2), 1–18 (2018)
26. Steunebrink, B.R., Dastani, M., Meyer, J.J.C., et al.: A logic of emotions for intelligent agents. In: Proceedings of the National Conference on Artificial Intelligence. vol. 22, p. 142. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999 (2007)
27. Steunebrink, B.R., Meyer, J.J.C., Dastani, M.: A formal model of emotions: Integrating qualitative and quantitative aspects. In: Dagstuhl Seminar Proceedings. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2008)
28. Vermeeren, A.P., Law, E.L.C., Roto, V., Obrist, M., Hoonhout, J., Väänänen-Vainio-Mattila, K.: User experience evaluation methods: current state and development needs. In: Proceedings of the 6th Nordic conference on human-computer interaction: Extending boundaries. pp. 521–530 (2010)
29. Zhao, Y., Borovikov, I., de Mesentier Silva, F., Beirami, A., Rupert, J., Somers, C., Harder, J., Kolen, J., Pinto, J., Pourabolghasem, R., et al.: Winning is not everything: Enhancing game development with intelligent agents. IEEE Transactions on Games **12**(2), 199–212 (2020)
30. Zook, A., Riedl, M.: A temporal data-driven player model for dynamic difficulty adjustment. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. vol. 8 (2012)

18

ANNEX A3

Automatic Evaluation and Ordering of Level Difficulty in Games

Miguel Reis
Instituto Superior Técnico
Lisboa, Portugal
miguel.o.reis@tecnico.ulisboa.pt

Rui Prada
INESC-ID
Instituto Superior Técnico
Lisboa, Portugal
rui.prada@tecnico.ulisboa.pt

Manuel Lopes
INESC-ID
Instituto Superior Técnico
Lisboa, Portugal
manuel.lopes@tecnico.ulisboa.pt

***Abstract*—In this work we propose a method to automatically order a set of game levels by their difficulty. We introduce two different complementary forms to evaluate difficulty that consider how easy it is to finish a level even committing some errors, and how hard it is to learn. Difficulty can be seen as a robustness to typical errors, or how much exploration and trials we need to learn how to solve the level. Our approach is based on first automatically create an agent to play a game level using the different forms of possible. Then, simulate the agents while considering some execution errors. Then, with the measures of difficulty, based on the robustness to errors, allows to order the different levels. Results show that the different measures of difficulty, and ways to simulate users' errors, are not equivalent (even if positively correlated) as they do not provide the same ordering.**

I. Introduction

Video games and interactive software in general need complex quality assurance mechanisms to ensure they are bug free and fulfill their functional requirements. In games, and other interactive software, it is also needed to evaluate the usability and consider human factors to determine if the game is, or not, enjoyable to play.

Testing involves substantial human labour, usually done by the developers themselves and playtesters. The first case can help but it does not solve all the problems as the developers are too close to the project. The second case is the traditional way of testing games, but has high costs and delays the development of the game, it is a bottleneck for the agile process that game development usually follows. Automation of the testing process of interactive software and games is, therefore, a prominent area of research.

One of the typical design concerns that needs testing, is the overall progression of difficulty of the game to maintain flow. This often means to define the best sequence of a set of levels. The sequence of levels can make or break how much enjoyment players get out of the game. For example, if a game starts with a level that is of high difficulty in which players need to have a good understanding of the mechanics to be successful, players can easily become frustrated. That is why most games include a tutorial [1] or start with very basic levels for a player to get understanding of the mechanics. A good pacing of the difficulty helps the players' learning the and mastering of the game.

Also, during game testing, it is important to be able to check if some change in the game parameters, e.g. jump height or friction, changes the overall progression of the game.

This paper describes our approach to automatically assess the difficulty of a game level and to order a set of levels by difficulty of a platform game. The approach fist determines a solution for the level using a search based algorithm. We then introduce three different ways to model players' errors during game execution. The measures of difficulty consider both the difficulty in learning, i.e. how much exploration is needed to find the best solution, and robustness to errors, i.e. how probable it is to finish the level even when committing an error. We are then able to generate traces of executions with errors that allow to measure the difficulty. With these measures we can order a set of levels.

We showcase our approach in a typical platform game but it can be generally applied in motor skill based games. Results show that we can indeed validate levels and order them. The different difficulty metrics do not provide the same ordering of levels and are thus complementary in use. They are positively correlated nevertheless.

II. Related Work

This section explores some work that has been explored in the main subject that this paper touches on.

A. Difficulty in Games

A very important and often overlooked in game testing and game design is level difficulty. This is due to the fact that a level difficulty in most cases cannot be perceived by directly looking at the level. You need to have a understanding of the mechanics of the game to be able to measure it and even a person who has it may not be able to correctly distinguish which levels are harder/easier than others. It also depends on what is difficult to each player. This task, often done by developers, is very time consuming because they usually have to playtest the level one by one several times before reaching a conclusion on how to order them.

With the goal in mind of defining difficulty, Maria-Virginia Aponte *et al.* [2] [3] presents a paper where they build a simple Pacman-like game where only one ghost is chasing Pacman while both characters were controlled using a A* pathfinding algorithm. The Pacman characters challenge was to eat as many pellets as it could without being killed by the ghost, while the ghosts challenge was to chase the player using the shortest path. The player AI used a Markov Decision Process using reinforcement learning with a Q-Learning Algorithm and to evaluate the difficulty, the metric chosen was the players movement speed. This value was an integer that could range from 0 to 7 and changed how frequently the Pacman could move, with 0 being the Pacman could move every 14 frames while with at 7 it would every 7 frames. The ghost would move always each 14 frames. The results were obtained after having played 45000 games for the AI to develop his policy in which then the mean score, based on how many pellets where eaten, of 5000 games was calculated. It could be seen that as the players speed went up, the score also started growing, although when getting closer to the player AI having almost double the movement speed of the baseline value, the score grew massively. The authors concluded then that is possible to evaluate a difficulty curve for a given game but that the experiment done could not be applied to every game and thus tried to find a more general definition that could be applied. They demonstrated a way of representing a challenge as a automaton where you have a starting state ”Not Started” which represents as a challenge not being started which leads to a state ”In Progress” in which the challenge is being performed which then can lead to either the ”Win” or ”Lose” state that represent the player passing or failing the challenge, respectively. With this they proposed a definition of a difficulty which was that the difficulty of a challenge given a time t is a conditional probability of losing the challenge before t considering all the challenges that have been solved before. The authors then talk about how a players’ abilities in games (e.g. aiming in shooter games) and how the better/more abilities the player has, the easier the challenges are to complete. They define that the relation between completing a challenge and a players ability is a conditional probability and so to test this, they proposed to measure a certain players ability on a given challenge. After doing many of these measures they were able to calculate the probability of a player winning the challenge, knowing his level for a particular ability. The authors explain that this can reveal important aspects of gameplay such as which abilities are more important for certain levels. Concluding, the authors affirm that game designers lack of methodology and tools to properly measure difficulty in games.

With a similar thought in mind, F. J. Gallego-Durán *et al.* tries to propose a new definition for measuring difficulty in [4]. The authors talk about how essential it is for a person to be in the ”Flow Channel” when performing a activity, this concept being that the more skills we have, the more challenging the activity should be on the risk we become bored of it, or in the other spectrum we become too anxious. But to do so it is necessary to evaluate the difficulty of the activity and the persons’ abilities. And so, the authors try to focus on the latter and aim to propose a new definition for difficulty, a way to measure it and then test the proposal in a practical case. The authors define ”difficulty as a cost: in order to successfully finish an activity, any learner has to pay a cost in time and effort”. With that effort is considered to be indirectly related to progress per time unit and therefore an activity is considered more difficult the less progress is done per time unit. In that work, difficulty is set to have the following properties, it is a continuous non-strictly decreasing function where the value ranges from 0 to 1. Due to these selected properties, the activities which we are trying to measure need to also follow certain guidelines such as activities require progress to be measurable, Score/Progress has to be non-strictly increasing function over time, activities must have a measurable success status or at least a maximum score and activities must be considered over time, which means several points of evaluation. With all these properties and guidelines properly defined, the authors present a mathematical definition of Difficulty where Difficulty over time is ”a value depending on all previous history of the i-th realization of an activity α by a learner l”. In order to test this definition the authors used a custom-made automated adaptive learning system called the ”PLMan learning system”. This system has 2 components, a game and a web application. The web application implements the learning system itself and lets students access their progress status, select level difficulty , get new activities assigned and upload their solutions which are automatically evaluated while the teachers are able to monitor the students work. The activities are based on the other component of the learning system, the game ”PLMan” which is a custom-developed game aimed at teaching Logic Programming and Reasoning. It is a Pac-man style game where the students use the Prolog programming language to control the character by creating a controller which then is simulated in the game. The game is automatically assessed throughout its score which is defined by the percentage of dots the character eats. A student can send as many controllers as he wants where each attempt was logged and as soon as he reaches a score of 75%, it unlocks a new map which were given difficulty levels by the teachers. The authors then proceed to show the results of the tests on three maps where the difficulty obtain trough the mathematical function corroborated with the difficulty given by the teachers. The authors then state 200 maps like those presented already have enough data to be analyzed and that many maps have already been re-classified after the teachers were able to identify some problems of the maps. They claim that this is a first step which will allow the framework in the future to automatically analyze the maps to identify problems and classify them. In conclusion the authors state that even though their definition of difficulty has some limitations, particularly in the properties that the activities must have, it also shows great advantages as it is a definition which is drawable which allows to show progress over time, it lets the teachers identify the most troublesome parts of learning activities and that different activities are easily comparable.

Another paper that relates to this topic, written by Rina R. Wehbe *et al.* [5], explores how design decisions affect the difficulty of platformer games. They start by mentioning that there are few guidelines and parameters that help game designers in the assessment of a levels difficulty because of the lack of understanding of what makes levels difficult in platformers. The authors then propose an evaluation where they change several parameters of a game and then test them using real players. The parameters that were tampered with for testing were the scroll speed of the controlled character, the size of a target a player wanted to hit, jump task complexity where single, double, triple jumps where tested where additional jumps had a extra target to hit and perspective(horizontal, vertical and z scrolling). The authors then recorded the perceived difficulty rating for each condition by the players and also captured in-game events such as the start of the jump, continuation of a complex jump combination and errors, distinguished by their cause. Using this, they tested 4 hypothesis, the first saying that increasing scroll speed would also increase the difficulty, second that decreasing the target size would increase difficulty, third that increasing jump complexity would increase difficulty and the fourth that the scroll perspective would not statistically affect the level difficulty. All the parameters were set to baseline values to obtain a baseline measure so it could be compared. After obtaining the results the authors concluded that the first and second hypotheses could be confirmed and that player scroll speed increase and target size decrease would make a level more difficult as per the results obtained in the perceived difficulty of the players and the in-game events. The 3rd hypothesis was only partly confirmed with the results because although there was a clear difference in difficulty from a single jump to a double jump, the same thing could not be said from the data recorded between the double jump and triple jump and in fact the data suggested that triple jump is at least as easy as double jump. The authors theorize that this happens due to the notion that repeated key sequences become easier because players get into a rhythm . However the forth hypothesis was disproved when analysing the data where vertical and z scrolling were statistically more difficult than horizontal scrolling but no more difficult than one another. The authors concluded then all the parameters tested influenced difficulty in a way and that these results can have an implication for the understanding of how game design decisions affect difficulty and the players experience.

Another approach to the problems of Difficulty in Games is to use Dynamic Difficulty Adjustment (DDA) systems that adapt the level difficulty to the player which is what S. Demediuk *et al.* [6] explore. The authors propose that by comparing a players performance to a DDA artificial opponent they are able to measure the skill level of the player. In order to measure this they implement a framework that first lets the players play against a adaptive opponent (ROSAS) in a 2D fighting game where the agents options at every step as well as their heuristic value are recorded. After this step the heuristic value of the actions chosen by the agent are averaged in which they call the HVA value. The bigger this value is, the closest to the optimal solution the agent is performing. By then obtaining this value of a fixed agent (e.g. Easy,Medium, Hard bot) using a fight with the adaptive one and by comparing it to the value the players got, the author claim that they are able to determine a players skill level. In order to test their approach they took 4 bots which were previously used in a competition and using ten fights calculated their HVA value and they were compared to Trueskill which is a ”ranking system used for competitive video games” which is able to measure a player skill level by having them play with another player. In the next step of the experiment, 5 human players played the ROSAS agent where their HVA value plus some observations about their skill were recorded. In these results the authors saw a great disparity between the highest and the lowest rated player in their HVA value which coincided with the observations recorded. Although the authors weren't able to rank them according to Trueskill by playing against each other, they believe that HVA is a ”promising method for ranking human players” but note that it doesn't give the best indication of the difference between players skill.

Another approach using Dynamic Difficulty Adjustment is used in a article by J. Hagelback *et al.* [7] where a study is done on players experience against static and dynamic artificial opponents. In this article, the authors set out with the premise that winning is not what gives the player the most pleasure, instead playing the game itself is. To confirm this they conducted an experiment on 60 people where each played a short session against one of five bots in a RTS game, 2 being static and 3 that could adapt during the game. The two static bot were set at a easy and medium difficulty while the adaptive started at a higher level but one had a small learning rate, one had a big learning rate and the other when the player had few units its difficulty was greatly reduced in order to let the player always win where the adaptive bots algorithm goal was for the game to be even. After playing against one of the bots, the players were given a questionnaire where you had to fill out some general question, such as Name, age, gender, result of the match and also a word pair section where the authors tried to infer the experienced enjoyment of facing the bot and the experienced strength and variation of it. To do so the authors created 6 clusters of words,2 related to enjoyment,2 to strength and 2 to variation where one had positive feelings and the other negative. The players were then asked to cross one of seven boxes where on the extremes you had one positive and one negative feeling which were not related(e.g. one positive enjoyment related and one negative strength related). The results show that the adaptive bots had the best variation score except for the one adaptive who loses on purpose which the writers justify because even though it remained an even game until the player had less than 5 units, the players tend to remember the last parts of the game. The authors note that the the adaptive bots had a better enjoyment score except again for the one who lost on purpose. These results also show that the static bots were not balanced for the level of the players, where one was too hard to beat and the other too easy. The authors conclude then that a competitive and adaptable game is more enjoyable for the player than a static one.

B. Discussion

The studies talked about above show some of the work done in Difficulty in games that relate and helped when it came to build the solution proposed in this paper. When talking about this topic we saw that currently there are not very well set ways or guidelines in order to evaluate a games difficulty so we saw some papers that tried to provide one for that fact. We also saw some papers that demonstrated that a players skill and past experience in games greatly affects its degree of difficulty and also that most, if not all of the game design choices that developers make, also affect a game being harder/easier to complete. Finally we saw some approaches using Dynamic Difficulty Adjustment systems, one that helped determine the skill level of a player and another that aimed to prove that people enjoyed playing these types of adversaries more than static ones.

In this paper even though we take the concepts from the works mentioned such as the alteration of game dynamics as we saw in Maria-Virginia Aponte *et al.* work and the concept of difficulty as a function of a players score in a map as in F. J. Gallego-Durán *et al.* work, we take a different approach where we're not interested in giving a final value of difficulty in a map but instead taking several maps and ranking them accordingly to their difficulty.

III. Proposed Solution

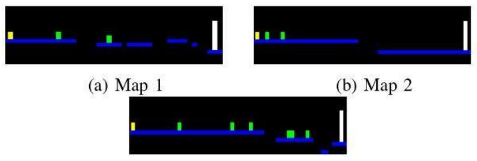
To test our approach we use levels of a platform game. We generate errors based on three different approaches and explore two different metrics to assess difficulty. Details are presented in the following sections.

A. Environment

The game selected was a platform like game that we built in order to be able to change several settings (map layout, player speed, jump parameters, etc). This game has a character which the player controls that can move sideways in either direction and jump. Additionally, the environment applies forces such as gravity and friction to the character, which influence its movement in the level. The player is tasked with reaching a flag which grants a bonus reward to the score.

If the player falls out of the map or if the time limit is reached we consider that the player has failed and in the case of the player falling out of the map we give a negative reward to its score. The score is calculated based on the player's position relative to the flag, the closest the better, and based on how much time has passed since the start of the map, the less the better. Levels are created in a file and every part of the map can be modified including players, goal and obstacle positions. The only obstacle present at the moment is a pipe but more obstacles and enemies can be easily added.

We also developed a level generator to create different maps to add diversity to the training and evaluation. These maps may vary in size, the jump holes vary in length and height between platforms, and obstacles are randomly put wherever there is ground. Figure 1 shows some examples of levels that can be generated.


(a) Map 1 (b) Map 2

(c) Map 3

Fig. 1: Example of 3 generated maps

B. Solver

The solver used in this work is based on the Learning Real Time A-Star (LRTA*) discussed in [8] and originally proposed in [9]. This solver works well for this case since we have a finite environment and small maps to explore. This algorithm is guaranteed to find a goal in these conditions.

The way it works is that initially we give it a problem it needs to solve. This problem includes the initial state, our goal, and the actions available in the environment. Initially the algorithm creates an empty table which will store the next state based on the current state and the action performed (table *result*) and creates another table which is used to store cost estimates indexed by state (table H).

The LRTA* agent receives the current state at each step of the environment execution. It checks each state in the H table. If not found it adds an new entry for it as well as for the *result* table for the state and each of the actions. As the agents is motivated to explore every state in the environment, it then checks if any entry in table *result* is null, meaning that (state, action) is not explored, and in this case it selects that action for execution. If in the current state, every action has been explored, the agent selects the one with the least LRTA*-Cost. After performing the action in the environment the agent updates the result table with the resultant state and the H table with least LRTA*-Cost for each action.

The LRTA*-Cost is a function which given a state, action, next state (based on action), checks if the next state as been explored, if not an estimate of the least cost it takes to reach the goal is computed based on heuristics. If the next state has been explored then it returns the value in the H table and adds the cost of the action.

The heuristic used for this game computes an approximation of how many steps would be required for the character to reach the goal based on its current position.

C. Error generation

To determine the difficulty, we simulate human error and induce it on the solution computed by the solver. The general idea is to check how those deviations from the optimal solution affect the performance of the agent, and use that information to predict difficulty.

We explored three different methods for the generation of errors:

- **Continuous random**
  Every time the agent chooses an action to perform in

the game, there is a 5% chance that it chooses a random action instead of the one given by the solver. The action do nothing is added to the set of actions.

- **Continuous on close keys** This simulates the mistake of pressing the wrong key, hence executing one action next, in the control scheme, to the one proposed by the solver. Every time the agent chooses an action to perform in the game, there is a 5% chance that it substitutes the action by one defined by a key proximity table. For example, the action close to the right key (that moves the character to the right), is either jumping or a key that does nothing.

- **Jump timing** This error simulates missing the perfect jump timing. To compute this, we identify "jump-zones" on the solution provided by the solver. These are positions close to where the agent is supposed to jump according to the solution provided by the solver. Once the agent enters a "jump-zone", a function is called to define where the agent will jump, taking in consideration that the closer the agent is to the original jump point the more probability the agent will have to jump there, as it can be seen in Figure 2. The actions the agent takes up until the jump occurs are always to move forward. The way we define the jump zone is by running the solution provided by solver and when we encounter a jump, we save the position the agent was 15 ticks before and set that as the starting point of the "jump-zone". The way we determine where the jump will occur is given by the probability mass function of a binomial distribution

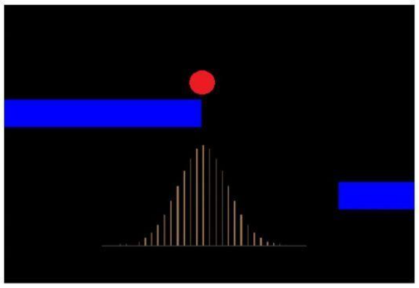


Fig. 2: Example of a jump when using the Jump Timing error generation. The red circle represents the original jump position while the graph below shows the distribution, relative to the position above, of where the jump will occur

We consider these three types of errors in order to cover the usual types of mistakes players make in a game. The first two relate to a "misclick" type of error where the player wanted to press a button but by mistake it clicks another. The third way we generate error simulates a perception error that happens when a player miss calculates where he should jump and presses the button either too late or too soon

D. *Measures of difficulty*

To assess the difficulty of a level, two metrics were used:

- **Probability of success**
  The Probability of success is the percentage of time that the agent is able to reach the goal after performing the level a few times, while applying the error.
- **Difficulty of learning**
  Difficulty of learning is a given value between 0 and 1 based on how many playtroughs of the map were needed for the solver to converge into a path.

We've chosen these metrics based on few assumptions. First, the main objective of the player is finishing the level by reaching the goal. A player looking at a map even if the person hasn't played the game, just by knowing the actions that are possible, can make a path that would look similar the the one obtained with the LRTA* algorithm. The problem is that players can make wrong judgments and even if they make the right ones can still make mechanical errors that can lead to failing the objective. In levels we low difficulty even if players make a mistake, they can recover and still finish the game, while in harder levels if players make a wrong decision or even fumble on the mechanical side (e.g. pressing the wrong button) it leads more easily to failing the objective.

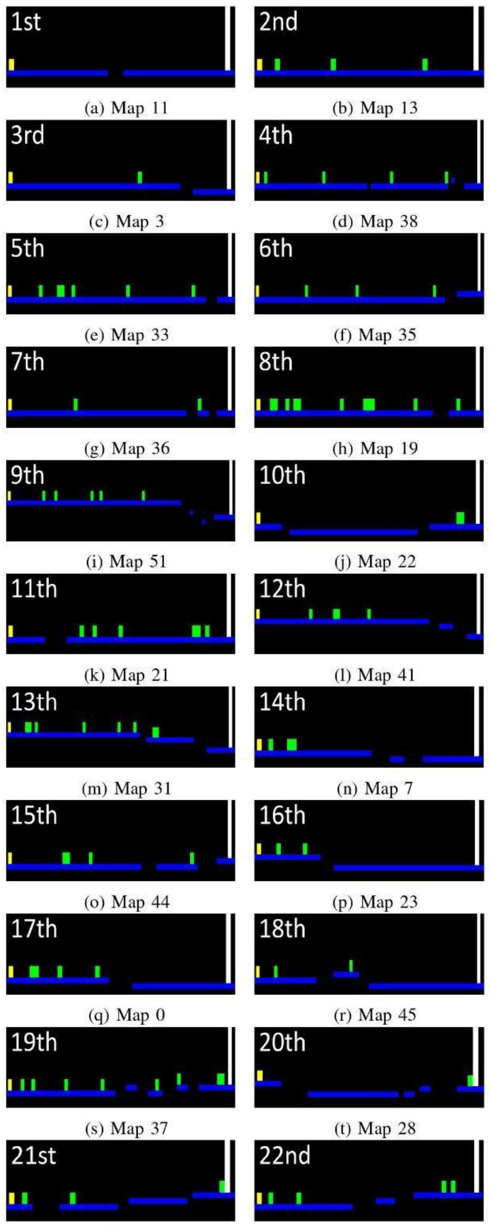
The other metric of Difficulty of learning which is based on how many trials it took for the agent to converge into the best solution can be correlated with the other metrics above to check if a game with more or longer paths can make the game harder for the player. We propose that the latter metric serves to help us understand why a map difficult and not exactly determine the difficulty of the map itself. Also, we note that the ways humans explore a level will be different from the one in LRTA*.

These metrics aligned with the specific game we used, but we believe that they can be applied to other games, as long as they an explicit goal state and explicit score at the end.

IV. Results

Using a tool that contained the elements mentioned in the section above we obtained the following results. We generated 60 maps using the map generator, some of which you can see in figure 3, then proceeded to train each of them with 3 different parameters. One where the world gravity had the default value, one with +10% gravity and another with -10% gravity. After training, we ran 2000 simulations of each map/parameter pairing for every type of error generation and recorded the results.

In order to analyze the different types of error generation, we took 10 random samples of the maps running at default gravity and ordered them according to the Probability of Success using the Continuous Random error generation.



(a) Map 11 (b) Map 13 (c) Map 3 (d) Map 38 (e) Map 33 (f) Map 35 (g) Map 36 (h) Map 19 (i) Map 51 (j) Map 22 (k) Map 21 (l) Map 41 (m) Map 31 (n) Map 7 (o) Map 44 (p) Map 23 (q) Map 0 (r) Map 45 (s) Map 37 (t) Map 28 (u) Map 5 (v) Map 26

Fig. 3: Figure contained the maps used ordered by Difficulty of Success using Timing error generation with ranking shown on top left corner in white

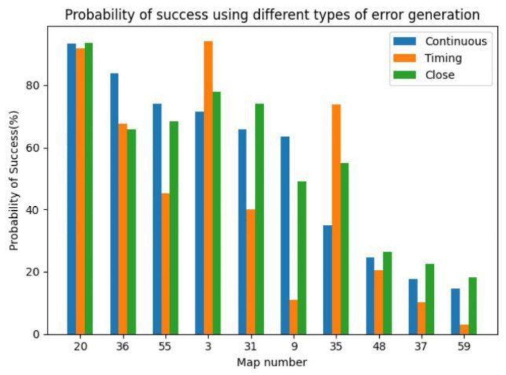


Fig. 4: Graphic showing the results of analyzing the Probability of Success using the different error generation types at default gravity

When looking at the results from Figure 4 we can see that the Continuous random and the Continuous on close keys are really similar on most maps with the latter performing just a little bit better which makes sense since these types of error generation are very alike except that the Continuous on close keys removes at each step the option furthest from the key to be pressed. When we look at the Timing error generation it usually generates worse performances than the other two although giving a similar ranking, which also makes sense since it emphasizes the errors on the jumps, which are the critical parts of the map. But on some other maps it generates better performance than the other two. When we look at the maps in question (3 and 35 in figures 3c and 3f, respectively) and analyze them we can see why. The reasoning of why the timing error generation gives such different results becomes more clear, since on map 3 it doesn't matter if the jump is mistimed because if the character keeps moving forward, the jump is not wide enough for him to fall. In map 35 this happens because the jump to the second platform happens on top of the pipe and if it delays a bit, the agent will fall to the ground right next to it and still make the jump to the second platform After looking at these maps, in particular map 3 when compared to map 20 which is ranked harder in the continuous type errors even though our playing experience tells us otherwise.

With this we reach the conclusion that the timing error generation does give us a better ranking of the difficulty between maps overall but that other 2 also offer a different perspective on what type of errors a human makes and that a combination of the two types can be further analyzed in the future.

To test out the different metrics proposed we selected 20 maps using random sampling and ranked them according to each metric. In figure 5 we can see these results which are ordered by the ranking of Probability of Success Looking

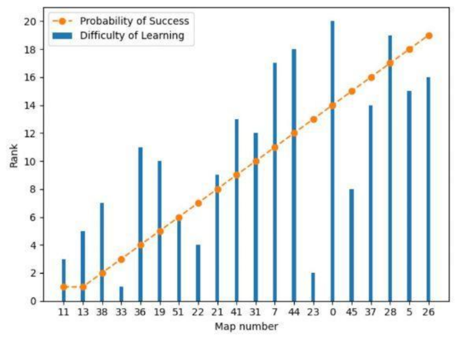


Fig. 5: Graphic showing the results of ranking 20 maps on the different metrics at default gravity, being 1 the easiest and 20 the hardest

at the figure above we can see that as a general trend that as the ranks for Probability of Success go up, the ranks for Difficulty of Learning also tend to get higher. However we do see some maps where the rankings don't correspond such as in maps 28 and 5 where map 5 is ranked just higher when comparing Probability of Success but much lower in Difficulty of Learning. When we look at the maps in figure 3t and 3u, respectively. At first sight it might seem that map 28 would be more difficult since it has more intricate jumps, but what the Probability of Success metric tell us otherwise. Furthermore we see that the difficulty of learning metric shows that map 5 is a more simple map of which makes us affirm that map 5 is a map which heavily relies on skill. This can be seen on further maps such as 26. We also see that the difficulty of learning still has some outliers, such as, in map 0 and map 23 (figures 3q, 3p) because it evaluates the complexity of the map in the optic of the algorithm used but we believe that a metric like this can help to contextualize the Difficulty of Success metric.

Next up we wanted to know the effect that the jump height difference and length had on the difficulty of the map using the Continuous Random error generation and the Probability of Success metric first at default gravity and then with +10% increased gravity to see how different a different dynamic would affect the correlation. We obtained the following results for maps that have 2 jumps which can be seen in Figure 6

As we can see from the top figure, the longer the jump is the more difficult the map becomes, as expected, but we can also check that, contrary to our expectations, the height difference between platforms in jumps does not seem to affect the difficulty by very much, if at all. We initially thought that this was due to the character in the game has a big jumping capability which makes the height less relevant but then after checking the results with increased gravity we notice that isn't the case since we see the same trend on the bottom figure.

We can conclude then that the jump height difference did not affect the difficulty of the maps tested. This may not be

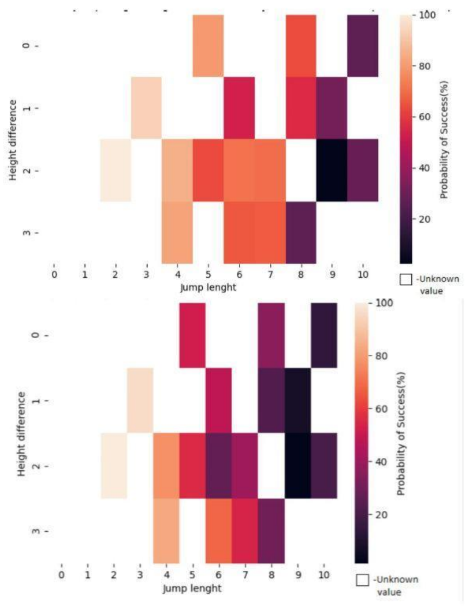


Fig. 6: Results showing correlation between jump height/length and difficulty. (Top) default gravity, (Bottom) Gravity increased by $+10\%$.

the case for every map since we set our maps had a set max height difference of 2 between platforms.

Finally, we wanted to check how the dynamics of the game itself can impact the difficulty of maps which can be seen in figure 7

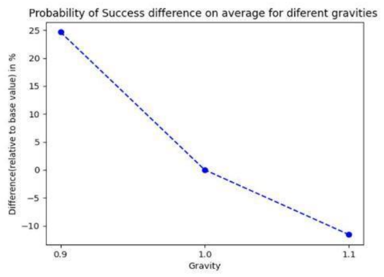


Fig. 7: Graphic showing the results of analyzing the Probability of Success using different gravity values with the Continuous Random error generation

This is a very straightforward result where if you increase the gravity our character will reach the goal less often on average and vice-versa, as expected. But this shows that tuning of the dynamic of the game itself such as gravity can greatly affect a level difficulty. This can be used in order to adjust the game to players of different skills. As mentioned in the introduction and in related works, maintaining a players interest is of the most importance, and so having options for players with higher skills and lower skills without having to change the map itself or create a new can be very useful.

V. Conclusions and Future Work

In this work we created a tool which can automatically analyze and order a large number of maps according to their difficulty. Our goal is to provide mechanisms to help level designers to test and sequence different levels.

In our approach we used a map generator to have a big sample size of maps that would be as varied as possible, we trained them using different values for gravity to understand how a different dynamic of the game would affect the difficulty then proceeded to use 3 different ways to generate errors while playing in order to better simulate human behavior while playing and recorded the results and evaluated using different metrics that helped us analyze the difficulty of the map and contextualize the ranking.

In the results we saw that by using different types of errors that people usually make, the ranking of levels changes. This allows to provide different ordering of the same levels based on the type of player each individual player does. We did however find that the timing error generation did produce rankings which better corresponded to our own personal rankings. We were also able to rank the maps properly using our Probability of Success metric and helped contextualize it with the Difficulty of Learning metric which helped us to understand what makes a map difficult. Finally we were able to understand how the layout of the map affected its difficulty by correlating the jump size/height difference and the difficulty which can help when trying to design maps with a difficulty in mind.

In future work we plan on adding/improving on the metrics presented that might give a better ranking on the maps used and help us better understand why the map is ranked that way and also develop our error generation methods in order to better simulate human behavior.

References

[1] E. Andersen, E. O'Rourke, Y.-E. Liu, R. Snider, J. Lowdermilk, D. Truong, S. Cooper, and Z. Popovic, "The impact of tutorials on games of varying complexity," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2012, pp. 59–68.

[2] M.-V. Aponte, G. Levieux, and S. Natkin, "Measuring the level of difficulty in single player video games," *Entertainment Computing*, vol. 2, no. 4, pp. 205–213, 2011.

[3] ——, "Scaling the level of difficulty in single player video games," in *Entertainment Computing – ICEC 2009*, S. Natkin and J. Dupire, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 24–35.

[4] F. J. Gallego-Durán, R. Molina-Carmona, and F. Llorens-Largo, "Measuring the difficulty of activities for adaptive learning," *Universal Access in the Information Society*, vol. 17, no. 2, pp. 335–348, 2018.

[5] R. R. Wehbe, E. D. Mekler, M. Schaekermann, E. Lank, and L. E. Nacke, "Testing incremental difficulty design in platformer games," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ser. CHI '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 5109–5113. [Online]. Available: https://doi.org/10.1145/3025453.3025697

[6] S. Demediuk, M. Tamassia, W. L. Raffe, F. Zambetta, F. F. Mueller, and X. Li, "Measuring player skill using dynamic difficulty adjustment," in *Proceedings of the Australasian Computer Science Week Multiconference*, 2018, pp. 1–7.

[7] J. Hagelback and S. J. Johansson, "Measuring player experience on runtime dynamic difficulty scaling in an rts game," in *2009 IEEE Symposium on Computational Intelligence and Games*. IEEE, 2009, pp. 46–52.

[8] P. N. Stuart Russell, *Artificial Intelligence: A Modern Approach*, 3rd ed., ser. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2010. [Online]. Available: http://gen.lib.rus.ec/book/index.php?md5=c37ce53726fb431e5815f9b1e573bfd6

[9] R. E. Korf, "Real-time heuristic search," *Artificial intelligence*, vol. 42, no. 2-3, pp. 189–211, 1990.

ANNEX A4

Validating the plot of Interactive Narrative games

Carolina Veloso
INESC-ID
Instituto Superior Técnico,
Universidade de Lisboa
Lisboa, Portugal
carolina.veloso@tecnico.ulisboa.pt

Rui Prada
INESC-ID
Instituto Superior Técnico,
Universidade de Lisboa
Lisboa, Portugal
rui.prada@tecnico.ulisboa.pt

***Abstract*—The authoring of interactive dialogues in video games is an overwhelming and complex task for game writers. Developing an Interactive Narrative that balances authorial intent and players' agency requires frequent in-depth testing. The limited range of tools to assist authors in verifying their story can limit the creation of more complex narratives. In this paper, we discuss the challenges of Interactive Story design and provide a model consisting of a set of metrics for testing interactive dialogues. Using this model, we developed a prototype for the *Story Validator* tool. This debugging tool allows game writers to experiment with different hypotheses and narrative properties in order to identify inconsistencies in the authored narrative and predict the output of different playthroughs with visual representation support. We conducted a series of user tests, Using the *Story Validator*, to investigate whether the tool adequately helps users identify problems that appear in the game's story. The results showed that the tool enables content creators to easily test their stories, setting our model as a good step towards automated verification for assistance of authoring interactive narratives.**

***Index Terms*—Interactive Narratives, Authoring Systems, Verification, Automated Playtesting**

I. Introduction

The presence of an immersive story in video games is often a central part of game experience and, consequently, a concern of design [1] [2]. Such games combine interactivity and storytelling as an Interactive Narrative (IN), often nonlinear, that allows the player's actions to alter the course of the story [3] [4]. At the core, IN functions similarly to the Choose-Your-Own-Adventure storybooks [5], where the reader is faced with various decision points at which they must make a choice, branching the story in different directions, often leading to different outcomes.

Deploying IN in games has enormous potential for enabling the creation of interactive systems that combine player interaction and dynamic plots, however, authors face many challenges when writing this type of stories [6]. Developing an IN where players can feel immersed and engaged involves making the player's actions and choices have a powerful influence on the narrative's direction, making it challenging for the author to guarantee a well-formed story. Each branch in the story has to be carefully tailored by the game's writer, who consequentially has to predict the different possible player's behaviours that can affect the story at various states so that they can present those choices to the player.

Most traditional approaches rely on extensive and rigorous playtesting to obtain information on how the players experience the narrative and what might need improvement. Playtesting provides insightful information that is not anticipated during development, helping authors ensure that both the game's narrative and the player's behaviour are well-balanced [7] [8]. However, obtaining quality feedback for a detailed analysis can be challenging, expensive, and time-consuming [9]. Various works have offered different solutions to handle narrative conflicts caused by user behaviour in-game. However, hardly any works have focused on letting the author simulate and question their narrative during development. Instead, they opt for online AI approaches (such as drama managers [10]) that, during gameplay, provide ways to dynamically adapt the narrative and resolve conflicts created by unintended player's actions. These approaches create changes to the narrative that the author might not have intended, leading them to lose control over their story.

On the other hand, most authoring tools, such as Twine [11] and Ren'Py [12], while they facilitate the creation of IN, they lack the proper tools to identify possible continuity errors, to keep track of specific narrative properties and to envision the output of playthroughs prior to human playtesting. Instead, these tools rely heavily on the author's intuition to foresee these challenges or on an exhaustive number of later playtests.

This work outlines the development of a tool that supports game writers in creating IN whilst maintaining the human author's directorial control. Essentially, we strive to maintain the idea of authorial intent [13] and develop a system that allows human authors to express their artistic intentions without feeling constrained. As a product of this work, we have designed, developed, and tested a prototype for the *Story Validator*, which traverses all possible narrative paths of a IN and provides insightful data on different story metrics and design issues encountered via visual representations.

This work was supported by the iv4XR (Intelligent Verification/Validation for Extended Reality Based Systems) project, EU H2020-ICT-2018-2020/856716, and by the National Funds Through FCT, Fundação para a Ciência e a Tecnologia, under project UIDB/50021/2020.

978-1-6654-3886-5/21/$31.00 ©2021 IEEE

II. Authoring Interactive Narratives

The authoring of interactive stories in video games is an overwhelming and complex task for game writers, both in narrative design and implementation. We identified four main challenges pertaining to the development of IN:

1) **Authorial intent vs player agency.** One of the most challenging problems with IN is the necessity to balance authorial intent and player agency in the context of storytelling [14]. According to Riedl, authorial intent is the trajectory that the game writer wishes the player to follow, regardless of how the player acts during the game. Because IN allows the user to interact freely within the story world, users often have the power to behave in ways that are inconsistent with the plot. This can either prevent the plot from advancing or make the player experience the story in ways that the creator did not intend.
2) **Narrative exponential growth.** As the plot grows in complexity and the number of decision points increases, the authoring experience will often require substantial changes to keep the narrative coherent, which can become overwhelming for the author [15]. Not to mention that the impact a choice has may end up only revealing itself in future states of the story, which is not always easy to predict.
3) **Gameplay variables that affect the story.** Besides what choices the player makes, the story's development can be based on different variables and status that are updated throughout to game. A common example is the "karma systems", which consider the player's good and evil deeds and shape the world around them as they progress. In more complex narratives, game variables can become challenging to keep track of, and their consequences may only unravel in later states in the game, which is not always easy for the author to foresee during development.
4) **Measuring the impact on user's experience.** The player play-styles are reflected not only in how they interact with the narrative but also in how they expect the story to unfold [16]. Due to the complex nature of IN, it is difficult for the author to predict the player's behaviour and overall emotional experience. This is challenging to predict without prior human playtesting, as it is hard to ensure during development.

III. Literature Review

A. Interactive Drama

In order to maintain narrative coherency even when the player has freedom of choice, some past approaches have focused on the idea of integrating real-time Artificial Intelligence (AI) methods to shape the narrative, allowing players to interact with the game freely but, at the same time, making sure the narrative still follows a coherent structure. A popular example, first proposed by Bates [17], is the use of a drama manager. A Drama Manager (DM) is an omniscient agent that monitors the game world and guides the player's experience through the story by searching for possible future plot points based on the evaluation of the current game world while still allowing the players to interact freely.

For instance, one of the most famous examples of the implementation of a DM is the Mateas and Stern interactive drama Façade [18]. Façade uses its DM to monitor and update the simulation in real-time in response to text the user inputs by selecting the next story beat.

Like in Façade, Riedl et al. presented the prototype for INTALE [19], in which the agents are directed by a DM called Automated Story Director. The Automated Story Director handles user interactions by maintaining a script of expected events and planning out new narrative follow-ups to respond to the player's actions and achieve all concrete narrative goals pre-defined by the author.

The studies mentioned above show that online AI approaches, particularly drama managers, are a popular solution to guarantee narrative coherence while allowing the player to interact with the game freely. Unfortunately, there has been hardly any work done regarding off-line approaches that attempt to identify possible narrative problems during development and allow the author to validate their narrative with different story metrics and predict different playthroughs' output.

B. Authoring Tools

Since the authoring process remains one of the most significant challenges in the development of interactive stories, there is a need for tools, toolkits and authoring systems that assist game writers in creating their content. Authoring tools for IN provide different visual representations and structural elements that allow authors to write their creative works.

With this in mind, we explored some of the existing authoring tools currently available for creating IN and their limitations, namely Tinderbox [20], Ren'Py [12], FAtiMA Toolkit [21] and Twine [11]. Our findings revealed that these authoring systems lacked the proper tools to analyse important narrative metrics and identify possible continuity errors. While these tools were designed to facilitate the authoring process, they nevertheless rely heavily on the author's diligent and methodical eye to predict all gameplay scenarios and possible complications that might arise, making story creation a wearing, expensive, and time-consuming process.

Nonetheless, due to its wide adoption and easy to work architecture, we concluded that Twine was the most promising candidate for the incorporation of our model, which we will describe in the next section.

Twine is a hypertext-based authoring tool designed to facilitate the creation of interactive stories with branching narratives. The creation of games with Twine requires only two elements: *Passages* and *Links*. *Passage* is the Twine's terminology for the nodes on the story graph that players navigate through. Each *Passage* contains a block of text that is shown to the player when they reach that *Passage* during gameplay. Additionally, *Passages* can also possess one or more

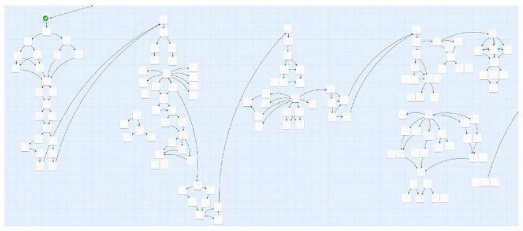


Fig. 1. A bird's-eye view of a storyboard in Twine.

tags, which function as labels that add information to the *Passage* but are not visible on the published version of the story.

Similar to branching narratives, where arcs connect nodes to each other, *Passages* are connected through *Links*, represented by an arrow on Twine's storyboard. To the player, these are displayed as points of interaction or decision points. The *Links* can be guarded by variables. Twine has a <<*if*>> macro, which is used to test the current value of the story's variables. These macros will influence the flow of the story since certain *Links* will only be available to the player if the conditions of the <<*if*>> macros are true.

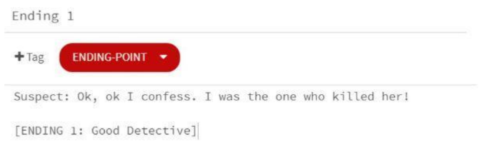
Because Twine is designed to develop and publish interactive stories on the Web, all its data is encoded in a single HTML file. However, while HTML is the default hypertext output for Twine, we found that exporting Twine's internal XML data in a JSON format would be an added value for reasons of simplicity and easier processing. This JSON file contains each *Passage*'s data, including the *Passages*' text, name, id, *Links*, and position on Twine's storyboard.

IV. A Model to Test Interactive Narratives

We sought out to develop a tool that supports game writers in the creation of IN in stages before human playtesting by letting the author explicitly test different hypotheses and narrative properties to identify possible design mistakes. To be useful to authors, such a tool needs to provide insightful data on different story metrics and identify design issues that a player could encounter during gameplay.

We take the assumption that the story space can be represented by a traversable graph with nodes (*Passages*) and edges (*Links*) that can be guarded by variables, and that the tool uses a Depth-First Search (DFS) based algorithm to traverse all possible narrative paths to gather data. With this in mind, we defined the following as useful narrative metrics:

- **Number of paths.** Enumeration of all possible traversals of the story graph, including the *Passages* the player visits on each narrative path. As a result with get the number of visits to each *Passage* as well and get an idea about which parts of the story are probably more often experienced by the players. When searching the graph, the algorithm begins at the starting node (obtained from the JSON data) and explores one *Link* at a time, adding each *Passage* visited to a stack. When it reaches an ending, the total number of paths is increased by one, the elements in the stack are saved as a path sequence, and we begin popping nodes from the stack until we find a node with an unvisited *Link*.
- **Endings Hit Percentage.** At the end of the game, the story reaches different endings, depending on the players' choices and state variables. The distribution of the endings players reach is often a concern of the design. As the search algorithm traverses all the paths, it keeps count of how many paths reach each ending. It then calculates the corresponding hit percentage. To ensure that, the story ending nodes need to to be identified. For example, by using the Twine tag system and adding an ENDING-POINT tag to the *Passages* that are defined as an ending (see Fig. 2).



Ending 1

+Tag ENDING-POINT

Suspect: Ok, ok I confess. I was the one who killed her!

[ENDING 1: Good Detective]

Fig. 2. Example of an ENDING-POINT tag in Twine.

- **Stroke Points.** Whenever the search algorithm reaches an ending, we compare the current path sequence with the previous one, intersecting their values to find common nodes in all paths. Eventually, we can identify which *Passages* are visited in all narrative paths. We named these *Passages*, Stroke Points. These can be part of the designer goals, as they can be useful to ensure some parts of the story is always conveyed to the player. However, it may happen that the author does not want to withdraw the player's ability to choose, in which case Stroke Points may become a problem.
- **Lost Plot.** Identifies narrative sections that, although included in the story by the author, are never reached in an actual playthrough due to some design or implementation error, for example, the conditions set in the <<*if*>> macros are never met. Ideally, in a well-constructed interactive story, all *Passages* should be visited at least once, and all paths should reach an ending. As the algorithm traverses the story, it keeps track of the *Passages* visited. Eventually, it can identify the existence of *Passages* that are never reached by any possible narrative path. Additionally, the iteration of the search may end in Dead-End, as it stops once it reaches a node without any edges (i.e., a *Passage* without any possible *Links*). Normally, this node is expected to be an ending, but it can also be a Dead-End — a non-ending *Passage* that impedes the player from progressing. Therefore, if the end *Passage* does not have an ENDING-POINT tag, it is identified a Dead-End.
- **Evolution of Variables.** A branching narrative might

contain different variables that the author needs to monitor. As the algorithm visits each *Passage*, it keeps track and updates those variables. This is used in the tool to display the values of the variables in a given path and to trace a timeline of the variable variations through a given playthrough.

The above-mentioned metrics were defined based on our personal experience as developers of interactive stories using Twine and based on a reflection to address the authoring problems discussed in section II. The same challenges were later reported by the user study's participants in section VI-D, which corroborated our ideas.

V. Story Validator Prototype

The implementation of the *Story Validator* tool follows the architecture presented in Figure 3. It is able to load an *interactive story* (in JavaScript Object Notation (JSON) format) created using Twine's authoring environment. It creates a representation of the story as a branching tree that is traversed by a Depth-First Search (DPS) based algorithm. It uses a DFS algorithm to be greedy in the exploration to explore a given story path until the end, as a human player would experience the story (i.e., starting at the root *Passage* and traversing through several *Passages* until reaching an ending point). The information collected is presented to the user through the use of a graphical interface (GUI), depicted in in Figure 4.

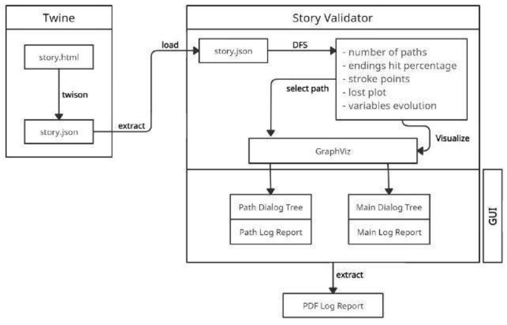


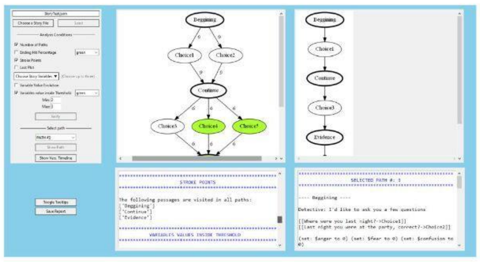
Fig. 3. The general architecture of the Story Validator tool.

The prototype was built in Python 3 using tkinter for the Graphical User Interface (GUI)[1]. The tool's GUI was developed following the conceptual organisation presented in Figure 5[2].

Using the *Story Validator's* GUI, the user selects (1) a JSON story file to be analysed by the tool. For this analysis, the user picks one or more options from a selection of analysis conditions (2), based on the metrics defined in section IV

[1]The source code can be found at https://github.com/iv4xr-project/in-story-validator

[2]Note that elements 4, 9 and 10 were added only after Phase 1 of user testing.



```
********************************************
                STROKE POINTS
********************************************

The following passages are visited in all paths:
['Beggining']
['Continue']
['Evidence']

********************************************
        VARIABLES VALUES INSIDE THRESHOLD
********************************************
```

```
********************************************
             SELECTED PATH #: 3
********************************************

---- Beggining ----

Detective: I'd like to ask you a few questions

[[Where were you last night?->Choice1]]
[[Last night you were at the party, correct?->Choice2]]

(set: $anger to 0) (set: $fear to 0) (set: $confusion to 0)
```

Fig. 4. The Story Validator prototype GUI.

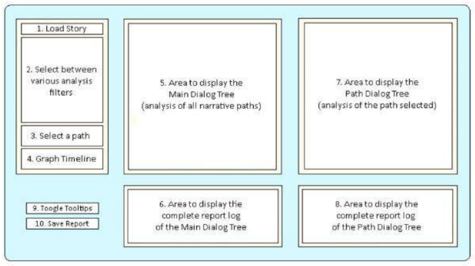
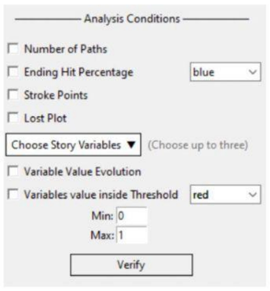

1. Load Story
2. Select between various analysis filters
3. Select a path
4. Graph Timeline
5. Area to display the Main Dialog Tree (analysis of all narrative paths)
7. Area to display the Path Dialog Tree (analysis of the path selected)
9. Toggle Tooltips
10. Save Report
6. Area to display the complete report log of the Main Dialog Tree
8. Area to display the complete report log of the Path Dialog Tree

Fig. 5. The Story Validator GUI conceptual structure.

(see Fig. 6). Each one of these conditions will influence different visual aspects of the Dialog Trees (5 and 7) and what information is shown on the Log Reports (6 and 8).



Analysis Conditions

- ☐ Number of Paths
- ☐ Ending Hit Percentage: blue
- ☐ Stroke Points
- ☐ Lost Plot
- Choose Story Variables ▼ (Choose up to three)
- ☐ Variable Value Evolution
- ☐ Variables value inside Threshold: red
- Min: 0
- Max: 1
- Verify

Fig. 6. The panel of analysis conditions.

Additionally, the user can select a story path to analyse in detail which will appear on the Path Dialog Tree (7) area.

As stated previously, this work's main objective is to support game writers by evaluating their game's narrative and working as a debugging tool. Next we will present how we envision authors using our prototype tool to identify and solve problems present their IN.

- **Keeping track of Passages visited.** Often during the creation of IN, there is the need to add, change and remove *Passages* and this process can quickly impact the

dynamic of the story. However, this is not always easy to predict as the impact of a *Passage* can reveal itself only in future states.
Furthermore, a branching narrative will often have several narrative paths that the player can take, resulting from their choices. For that reason, as the number of possible narrative paths grows, it becomes tougher for the author to keep track of the *Passages* that are visited.
The tool's algorithm runs through the story by selecting different options until it reaches an ending. Then it starts a new playthrough and repeats, choosing other options. In the end, the Main Dialog Tree (5) displays a tree with all the narrative paths that the player can take, giving the author a general idea of how the story flows. In order to make a more in-depth assessment, the Main Report Log (6) displays which *Passages* are visited in each path. Additionally, the user can pick a path to analyse in detail (3), which is displayed on the Path Dialog Tree (7).

- **Keep track of endings' reachability.** Depending on the choices in dialogue made, the player is led to different endings. However, it is difficult for the author to predict and monitor the endings' distribution without it being a laborious and time-consuming task. As a design objective, the author may want to create certain restrictions on the distribution of the endings, such as having an ending that is more common to obtain or an ending with only one possible path. If the analysis condition *Endings Hit Percentage* is selected, the Main Report Log (6) then provides the author with percentages on the likelihood of reaching each ending. Besides, on the Main Dialog Tree (5), the user can observe the paths' distribution if the analysis condition *Number of Paths* is selected.
- **Keep track of the story's variables.** In IN, the path in which the narrative develops can be dependent on different game's variables. Having variables updating depending on the characters or plot state can make the interactive story more interesting for the player; however, it is difficult for the author to keep track and control those values over multiple interactions. The tool can identify and keep track of all variables defined by the author and their values throughout each path. The user can also select which variables they wish to analyse (up to three). By selecting the analysis condition *Variables Value Evolution*, the user can observe the value changes of each variable. By selecting the analysis condition *Variables value inside Threshold*, the tool highlights the *Passages* where the variables have a value between the MIN and MAX values, both defined by the user. The story variables are also presented in a dotted chart (4), displaying the value's changes throughout the story.
- **Avoiding Dead-Ends and losing plot.** Dead-Ends typically happen due to an error in defining the *Passages* that should come afterwards, meaning they have "entering" conditions that are impossible to meet or if the path is an "endless" path. The designer must identify these cases, as they abruptly stop the player from continuing playing. However, doing so is difficult due to the combinatorial nature of the exploration of the story. Furthermore, it is also essential to identify if there are sections in the story that are never visited, at the risk of losing important parts of the plot. If the user selects the analysis condition *Lost Plot*, the Main Report Log (6) will print out the *Passages* that were never visited and display which paths were not able to reach an ending. Additionally, on the Main Dialog Tree (5), *Passages* that are never visited will appear as single nodes with no edges connected to them, and on the Dialog Trees (5 and 7), Dead-End nodes will take on a rectangular shape in oppose to its regular round shape, so they are easier to identify for the user.

A. *Emulating personas and playstyles*

As mentioned in section [II], the players' play-styles are reflected not only in how they act within the plot, but also on how they expect the story to respond to their actions.

The author may want to guarantee that some types of players follows a particular trajectory. Therefore, our intent was to develop an agent that simulates various playthroughs of the game, emulating the behaviour of different players, in order to predict user experience and study the story's output.

In their work, Stahlke et al. [23] developed a framework, named PathOS, that simulates human testing sessions using intelligent agents. These agents mimic the behaviour of human players by following a set of heuristics (e.g., curiosity, aggression, and completion), that are configured to reflect different play-styles. Our work proposes a similar approach, by letting the author assign different weights to each of the story variables to create different personas that represent different playstyles. These personas will have a tendency to favour options in the story that have stronger impact in certain variables.

For this, we use an informed **Greedy search** (see Algorithm [1]) using as heuristic based on the defined weights for the "persona" under test. Each story has a set of variables $V = (v_1, v_2, ..., v_n)$, where *n* is the total of story variables, and to each of these variables *V*, the writer assigns weights $W = (w_1, w_2, ..., w_n)$. The heuristic then drives the playthrough of the story to give preferences to *Links* that raise the value of variables it gives higher weight.

For example, if we treat the variables as "emotional states", the author may study how the story unfolds given the behavior of a more aggressive type of player in their game scenario. They can give greater weight to an "angry" variable that depicts the emotion of a character in the story and, consequently, drive the algorithm to choose the options that lead to higher value to that emotion. By specifying different weights to each story variable (e.g., 75% fear, 15% confusion, 5% angry), the author creates agents with distinct preferences that play the game in different ways.

It is important to note that since the greedy algorithm does not consider the overall problem and always makes a locally-optimal choice at each step based on the information it has, it often does not produce the most optimal solution. However,

Algorithm 1: Greedy search using heuristic based on the defined weights.

```
Function Greedy_step(hightest_total, passage, V, W):
    /* gather all the passage's links */
    $links = passage.get("links")$
    for $l$ in $links$ do
        /* see that $v_{nl}$ is the value update */
        /* of the variable $n$ on link $l$ */
        $total_l = v_{1l} * w_1 + v_{2l} * w_2 + \ldots + v_{nl} * w_n$
    end
    /* chose link that maximizes the total value */
    /* assume $x$ is the total number of links */
    $chosen\_link = \mathbf{max}(total_{la}, total_{lb}, \ldots, total_{lx})$
    /* update highest total value */
    $highest\_total = highest\_total + chosen\_link$
    /* chose link with highest value to visit */
    $next\_passage = l_{max}$
    /* move to next iteration */
    $Greedy\_step(highest\_total, next\_passage, V, W)$
```

we need to remember that, in most cases, the human player is also unaware of the overall game-space during gameplay and, therefore, makes the choices they believe to be the most optimal at each step, based on the information they currently have and their player profile. Thus, we believe that a greedy algorithm best emulates the human behavior in this scenario.

VI. Experimental Results

The following section presents the methods and results of two user studies that were conducted with the intent of: finding out if the tool adequately helps the users identify problems in the game story, and determining whether users can operate the tool with ease and identify usability issues.

A. The Scenario

For the tests we used Twine's authoring tool to create an *interactive story*, which we named *"The Murder Case"*.

The story has the following setting: the player plays a detective tasked with solving a murder. The detective's main suspect is brought to the police station for questioning and, since existing evidence is not sufficient, the player will need a confession. However, there is a catch: the suspect NPC is a very wealthy man and upsetting him during questioning will lead him to ask for his lawyer and the detective to lose the case. Depending on the dialogue choices made when talking with the suspect, the player is led to different endings that reflect their crime-solving skills. To stipulate the effect the player's choices have in the narrative's unfolding, we attach (and update) variables to each dialogue option. These variables represent the different emotional states of the suspect NPC: *anger*, *confusion*, and *fear*.

The scenario contains 3 different endings ("Good detective", "Lawyered" and "Not enough evidence"), 14 *Passages* and 20 possible narrative paths players can take.

Additionally, we created an alternative version of the story with added issues to be used in the User Test 2. This version, named *"TMC - failed version"*, includes 2 *Passages* that are never visited and 3 Dead-Ends.

B. User Study 1

In the first study, we examined how the users feel about the tool's design and checked its usability. The study was conducted with 5 users (3 female and 2 male), ages between 18 and 24. Most participants admitted they played games either regularly (40%) or every day (40%), with some (60%) enjoying games with interactive stories and branching narratives. Out of the 5 participants, 3 stated they had knowledge in game development, manly working as a Game Programmer and/or Game Writer.

The participants were asked to perform 10 tasks using the *Story Validator* and *"The Murder Case"* scenario. These tasks included calculating the total number of paths, finding the endings' distribution, studying the story variables' evolution in each path, among others. This version only allowed users to test one story variable at a time.

While the participants completed the tasks, we observed their performance and took notes. We used the think-aloud method, where participants used the system while continuously verbalising their thoughts.

In general, the participants were able to complete the tasks, but some difficulties were found in the tasks that required checking the values of variables. For example, two participants did not complete a task that involved counting the number of paths that reached an ending with a variable "anger" = 0, while one was not able to find *Passages* in which the variable "anger" had values between -4 and -2.

The participants found the *Story Validator* tool straightforward and easy to use, giving it an average score of 83.5 (SD = 9.45) in the System Usability Scale (SUS). According to a study made by Bangor, Kortum, and Miller [22], these results suggest that tool has "passable" overall usability.

In the study, participants also proposed different suggestions regarding the tools' usability, for example:

1) **Help understanding how the tool works.** Some users noted that the quantity of features was a bit intimidating ("[the tool] is very overwhelming at first"). It was suggested some form of assistance to explain the tool's functionality ("Once you start clicking some checkboxes, you learn pretty quickly how it works [...] but having a help button or something similar would have helped a lot.").
2) **Easier to read log report.** In cases where multiple analysis conditions were selected, users had issues while navigating through the log report box and often had to keep scrolling up and down to locate what they were seeking. Participants noted that if the box was larger it would be easier to navigate.

3) **Ability to analyse more than one variable.** One of the users reported the desire to analyse more than one variable at a time. While this did not hinder their ability to complete the tasks, their suggestion was noted down.

Based on the feedback gathered, we improved the prototype and added some new features, including:

- The option to test more than one variable simultaneously.
- A "Toggle Tooltips" button, which allows the user to hover different elements on the tool's menu to display a short message detailing how each works.
- A "Print Report" button, which creates a full pdf report of the story analysis that the user can access outside the tool's workspace.
- A dotted chart graph, which displays a timeline with the changes in the values of each story variable select for analysis throughout the story path selected.

C. User Study 2

For the second user study we used an improved version of the *Story Validator* that addressed the usability issues found in the first study and included a few extra features.

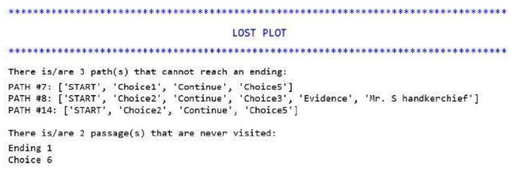
The goal of this study was to test if users were able to use the *Story Validator* to identify problems and various design issues in the interactive story (the *"TMC - failed version"*), and suggest possible solutions to the problems they find.

We conducted the test with 20 participants (8 female and 12 male). Except for one participant, all others admitted they played video games either every day (35%) or regularly (60%). Nine of the twenty participants said to have some knowledge in game development (as Game Programmers, Game Designers or Game Writers). Five of those participants were familiar with Twine's authoring tool prior to the test and considered themselves to have "Intermediate" expertise.

The scenario that participants tested had the following issues:

- **Problem 1:** The Path #7 does not reach an Ending.
- **Problem 2:** The Path #8 does not reach an Ending.
- **Problem 3:** The Path #14 does not reach an Ending.
- **Problem 4:** The Passage "Ending 1" is never visited.
- **Problem 5:** The Passage "Choice 6" is never visited.

All the previous problems can be identified in the **Story Validator** under the Analysis Condition *"Lost Plot"*, which reveals *Passages* that can not be visited and Paths that do not reach an ending, if existent.



```
*******************************************************************************
                                   LOST PLOT
*******************************************************************************

There is/are 3 path(s) that cannot reach an ending:
PATH #7: ['START', 'Choice1', 'Continue', 'Choice5']
PATH #8: ['START', 'Choice2', 'Continue', 'Choice3', 'Evidence', 'Mr. S handkerchief']
PATH #14: ['START', 'Choice2', 'Continue', 'Choice5']

There is/are 2 passage(s) that are never visited:
Ending 1
Choice 6
```

Fig. 7. Analysis Condition "Lost Plot" for the "TMC - failed version".

However, the reasoning behind each of these problems is unique, and the user needs to make use of the tool in order to grasp what exactly is causing each of these issues.

For example, problems 1 and 3 have a similar cause: both reach a *Dead-End* at the *Passage* "Choice 5". Using the **Story Validator**, the user will realize that during the story's creation, the writer did not add any *Links* leaving *Passage* "Choice 5" and, consequently, the play-through stops abruptly there.

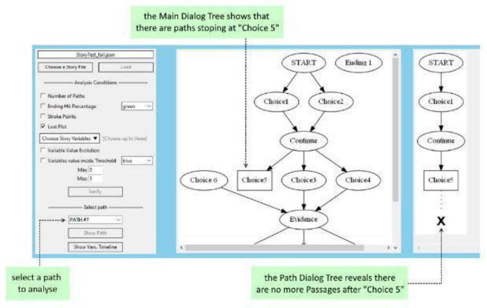


Fig. 8. Analysing problems 1 and 3.

In general, the participants were able to identify the problems, except for three who could not identify problem 2. We believe that the reason for the misidentifying of this problem was that instead of using the Analysis Condition *"Lost Plot"* to identify the issues within the story, they found them by observing the tree displaying the in tool directly. While this method is legitimate, none of these 3 participants were able to identify and solve problem 2.

Additionally, two users (10%) could not find a solution for problems 1 and 3. In turn, problem 5 seemed to be the one that was harder to solve for at least five of the participants (25%). For all the other problems, the completion rate is 100%.

According to the average participant, the task of identifying the problems within the narrative was considered relatively easier than trying to propose a solution. The results also show the time it took users to identify all the problems was 52.3 sec. (SD = 22.66), while proposing a solution took, on average, 146.2 sec. (SD = 31.84). Overall, the time values for both finding and solving problems prove that the tool is efficient and that, with a bit of experience with the tool, users can quickly use the tool to pinpoint and repair errors in their interactive stories.

In the end, participants gave the *Story Validator* tool an average SUS score of 92.4 (SD = 4.76). These results suggest that our system is considered a "truly superior product" [22].

D. Additional results

Before each experiment, we asked the participants, together with the demographics questionnaire, their preferred tool to write stories for games, and what problems they usually face in the task.

The most preferred authoring tool to write stories for games was a basic word processing tool (76%), such as Microsoft Word, Google Docs or LibreOffice Writer, but some had experience using Twine (24%). The users also reported the following problems when using their preferred authoring tool for writing IN:

- Difficulty keeping track of game variables and/or objects;
- Difficulty identifying "Dead-Ends", meaning moments in the game that prevent the player from continuing playing;
- Difficulty keeping track plot moments in the game that the player skips unintentionally, losing the plot coherence;
- The lack of ability to keep track of user experience (before playtesting).

These findings are aligned with the metrics we propose in section IV.

VII. Conclusion

In this paper, we have underlined some of the current challenges concerning the authoring process of IN. We addressed the lack of tools that provided ways for the authors to test their narrative while considering the player's agency, during the game's developmental stage, without requiring playtesting. More often than not, these works opt for online AI approaches that, during gameplay, dynamically adapt the narrative and resolve conflicts created by unintended player's actions. This might lead to situations where the system takes control of the story, replacing human authorship.

With this work, we set ourselves to develop a tool for testing interactive dialogues for video games. With such tool, we believe that authors gain some control that enables them to define more complex narratives to express their artistic intentions without feeling constrained. This approach has been designed to facilitate the development of IN in stages before human playtesting by letting the author explicitly test different hypotheses and narrative properties to identify possible design mistakes. The tool's GUI allows for a clearer picture of the IN authoring process, by providing a visual representation of the narrative structure through the use of tree structures, that run through different test conditions. The two studies we conducted show some good sign for the approach and tool proposed. As several participants confirmed ("I would definitely use this tool to test my stories", "[the tool] really helps give the designer an idea of how their narrative works", "using [the tool] would save game developers so much time") the tool proves to be an essential asset for the creation of IN.

Overall, we believe that, as a first approach to this type of systems, our prototype managed to achieve the proposed objectives.

Acknowledgments

We gratefully acknowledge the assistance and support provided by the colleagues of project iv4XR at INESC-ID. We would also like to thank all those who participated in the user studies; their feedback was extremely helpful.

References

[1] E. Aarseth, "A narrative theory of games," in Proceedings of the international conference on the foundations of digital Games, 2012, pp. 129–133.

[2] H. Qin, P.-L. Patrick Rau, and G. Salvendy, "Measuring player immersion in the computer game narrative," Intl. Journal of Human– Computer Interaction, vol. 25, no. 2, pp. 107–133, 2009.

[3] M. O. Riedl and V. Bulitko, "Interactive narrative: An intelligent systems approach," Ai Magazine, vol. 34, no. 1, pp. 67–67, 2013.

[4] S. Dinehart, "Dramatic play," 2009, online; Retrived 5-November-2020. [Online]. Available: http://www.gamasutra.com/view/feature/4061/dramaticplay.php

[5] B. Books, Choose Your Own Adventure Book Series. Bantam Books: NYC, 1979 - 1998.

[6] M. Mateas, "The authoring challenge in interactive storytelling," in Joint International Conference on Interactive Digital Storytelling. Springer, 2010

[7] P. Mirza-Babaei, N. Moosajee, and B. Drenikow, "Playtesting for indie studios," in Proceedings of the 20th International Academic Mindtrek Conference, 2016, pp. 366–374.

[8] P. Mirza-Babaei, V. Zammitto, J. Niesenhaus, M. Sangin, and L. Nacke, "Games user research: practice, methods, and applications," in CHI'13 Extended Abstracts on Human Factors in Computing Systems, 2013, pp. 3219–3222.

[9] N. Moosajee and P. Mirza-Babaei, "Games user research (gur) for indie studios," in Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems, 2016, pp. 3159– 3165.

[10] J. Bates, "Virtual reality, art, and entertainment," Presence: Teleoperators and Virtual Environments, vol. 1, no. 1, pp. 133–138, 1992.

[11] C. Klimas, "Twine - an open-source tool for telling interactive, nonlinear stories," 2020, online; Retrived 5-January-2020. [Online]. Available: https://twinery.org/

[12] T. Rothamel, "The ren'py visual novel engine," 2020, online; Retrieved 16-November-2020. [Online]. Available: https://www.renpy.org/

[13] M. O. Riedl, "Incorporating authorial intent into generative narrative systems." in AAAI Spring Symposium: Intelligent Narrative Technologies II, 2009, pp. 91–94.

[14] R. Aylett, "Emergent narrative, social immersion and "storification"," in Proceedings of the 1st International Workshop on Narrative and Interactive Learning Environments, 2000, pp. 35–44.

[15] E. Adams, Fundamentals of game design. Pearson Education, 2014, pp. 172–173.

[16] S. Dow, B. MacIntyre, and M. Mateas, "Styles of play in immersive and interactive story: case studies from a gallery installation of ar façade," in Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology, 2008, pp. 373–380

[17] J. Bates, "Virtual reality, art, and entertainment," Presence: Teleoperators and Virtual Environments, vol. 1, no. 1, pp. 133–138, 1992.

[18] Mateas, Michael, and Andrew Stern. "Façade: An experiment in building a fully-realized interactive drama." Game developers conference. Vol. 2. 2003.

[19] M. O. Riedl and A. Stern, "Believable agents and intelligent story adaptation for interactive storytelling," in International Conference on Technologies for Interactive Digital Storytelling and Entertainment. Springer, 2006, pp. 1–12.

[20] M. Bernstein, "Collage, composites, construction," in Proceedings of the fourteenth ACM conference on Hypertext and hypermedia, 2003, pp. 122–123.

[21] S. Mascarenhas, M. Guimarães, R. Prada, J. Dias, P. A. Santos, K. Star, B. Hirsh, E. Spice, and R. Kommeren, "A virtual agent toolkit for serious games developers," in 2018 IEEE Conference on Computational Intelligence and Games (CIG). IEEE, 2018, pp. 1–7.

[22] A. Bangor, P. T. Kortum, and J. T. Miller, "An empirical evaluation of the system usability scale," Intl. Journal of Human–Computer Interaction, vol. 24, no. 6, pp. 574–594, 2008.

[23] Stahlke, Samantha, Atiya Nova, and Pejman Mirza-Babaei. "Artificial Players in the Design Process: Developing an Automated Testing Tool for Game Level and World Design." Proceedings of the Annual Symposium on Computer-Human Interaction in Play. 2020.