Intelligent Verification/Validation for XR Based Systems

Research and Innovation Action

Grant agreement no.: 856716

D4.4 – Report describing SETAs

iv4XR – WP4 – D4.4

Version 1.8

December 2022

Project Reference	EU H2020-ICT-2018-3 - 856716
Due Date	31/12/2022
Actual Date	29/12/2022
Document Author/s	Pedro M. Fernandes (INESC-ID), Rui Prada (INESC-ID), Saba Ansari (UU), Marta Couto (INESC-ID), Manuel Lopes (INESC-ID), Carlos Martinho (INESC-ID), Wishnu Prasetya (UU), Victor Gabillon (THALES-SIX), Fitsum Kifetew (FBK), Frank Dignum (UMU)
Version	1.8
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	25/10/2022	Initial document structure and contents	PF
1.1	22/11/2022	Added OCC and Coverage Sections	SA
1.2	05/12/2022	Completed the Components of User Experience Section	PF
1.3	09/12/2022	Completed and reviewed sections 1-6	PF
1.4	13/12/2022	Added details on the CS sections and overall minor edits	RP
1.5	13/12/2022	Added details about the UX framework and the work on Cognitive Load work	MC
1.6	15/12/2022	Reduced the size of some sections and other minor edits	PF
1.7	28/12/2022	Added the executive summary	PF
1.8	29/12/2022	Final arrangements for submission	RP

Document Quality Control			
Version QA	**Date**	**Comments (and if appropriate reason for change)**	**Initials of QA Person**
1.6	20/12/2022	Review and minor changes	ML

Document Authors and Quality Assurance Checks		
Author Initials	**Name of Author**	**Institution**
RP	Rui Prada	INESC-ID
PF	Pedro M. Fernandes	INESC-ID
SA	Saba Ansari	Utrecht University (UU)
MC	Marta Couto	INESC-ID
ML	Manuel Lopes	INESC-ID

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This deliverable presents a summary of the Socio-Emotional Testing Agents (SETAs) that were developed as part of WP4. It begins by explaining the modular architecture of the SETAs and then presents the several components that were developed during the iv4XR project. These components pertain both to the behaviour of the SETAs as well as the automatic measurement of different components of user experience. At the end of the deliverable, we provide example usages of the SETAs and of how the different components can be used by designers to create tailored made user experience testing agents for their applications.

SECTION 1 - INTRODUCTION

In a world where agile software development is becoming the norm, which entails quick development sprints and a reliance on automatic testing, there is concern for the testing of User eXperience (UX) during development. This is especially relevant for complex systems like Augmented Reality (AR) and Virtual Reality (VR) applications. Currently, testing UX on such systems requires testing with humans, which is both money and time consuming. It becomes infeasible to test the implications to UX each time developers change something in the application as it would slow down the development process too much. However, having access to quick and on demand estimates of UX would allow developers to create better and more user friendly applications. It would also greatly benefit developers and environment designers to know what actions and paths users are most likely to take.
The creation of Socio-Emotional Testing Agents (SETAs) is our approach to begin solving these problems and allow UX concerns to be tested automatically during development, aiding the creation of better applications. SETAs are agents capable of interacting with an environment and providing estimates of different components of UX. Such capabilities allow them to be used by developers and testers to quickly and automatically have a better understanding of whether their UX design goals are being met or not. These agents could, for example, allow developers to automatically know which areas of an environment are not providing enough stimulus to users or which ones are too overwhelming. They could also give developers an idea of how different the experience would be to different users.

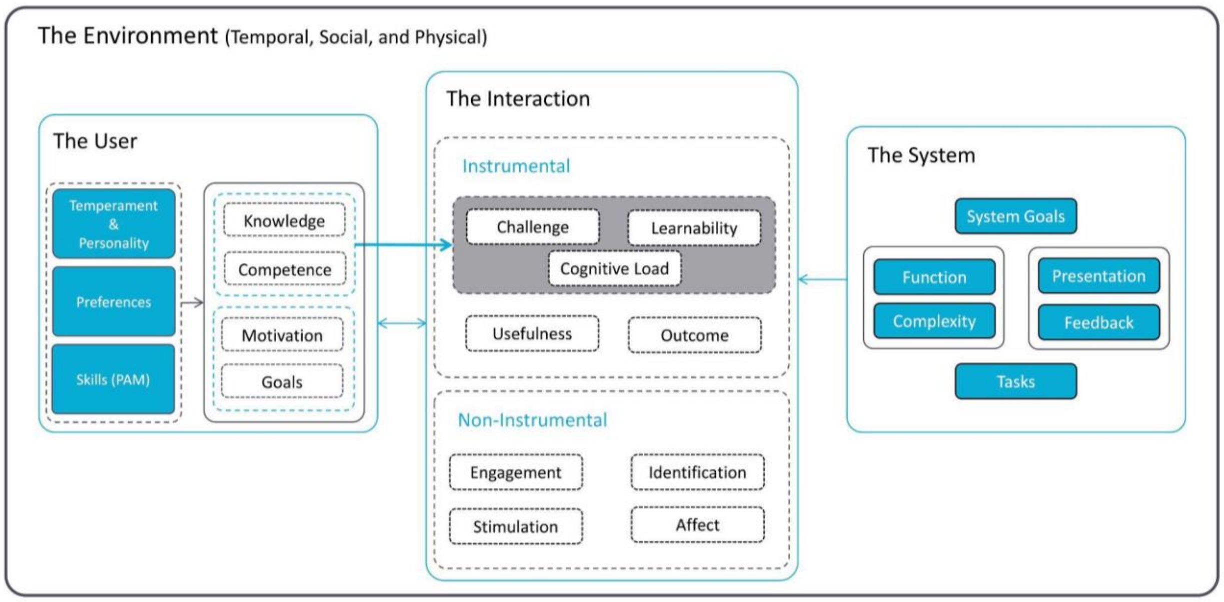


Figure 1: The different components of UX (the user interaction with the system in a given environment), and the variables contributing to User eXperience.

To have a comprehensive view of UX, we gathered previous definitions in literature and defined UX as emerging from a user's interaction with a product/service/software. UX is a consequence of prior experiences, attitudes, skills, habits, and personality. It is a subjective, situated, complex, and dynamic encounter.
To evaluate UX, we need to consider the user, the product/service/software, the context or environment, and the variables that emerge during the interaction itself. Figure 1 provides an overview of a UX framework we constructed, focused on automated evaluation of UX. To move from the current practices towards automated tests it is important to have clear definitions and a modular structure to help define test agents.
The interactions are situated i.e., they take place in a given context. For the environment we consider three layers: the temporal layer, the social layer and the physical layer. The temporal layer considers long-term interactions. Most products/services/software are not single use and so a good UX evaluation should consider multiple interactions over time. The social layer relates to individual UX experiences that are constructed in social interaction. Finally the physical context relates to physical aspects of the environment that can affect the interaction experience.
In the environment we see the user interacting with the system. We have user variables and system variables that come together when the user is using the system, for example executing a task, as this will produce the interaction variables. At the beginning of an interaction the user has a set of fixed variables (in blue) and variables that should change with the interaction. The interaction is divided into instrumental and non-instrumental variables. All variables are expected to change in each interaction. The last box represents the system variables which we consider to be fixed at the beginning of the interaction. These are variables of the system that will impact the interaction and consequently, the UX. The SETAs work developed within the lifetime of the project aims at covering some of the aspects illustrated in Figure 1.

SECTION 2 - THE MODULAR SETA

User experience (UX) is a broad, complex and multi-faceted concept. It encompasses many different components of human experience along with characteristics of the system itself and the surrounding environment (Figure 1). It is therefore unfeasible to have a single, unidimensional measure of UX. What can be done is the measurement or prediction of different components of UX according to what is most relevant to testers and designers. With this in mind, we have implemented our SETAs to be modular by design. This means that they can be run with different modules, each endowing them with the ability to predict different components of UX. They can also be easily expanded with new modules to allow them to predict novel components of UX as new predictors are created.
For example, to the designer of an aeroplane piloting simulation, the most relevant components of UX to be predicted might be cognitive load and the level of arousal. It might be of little relevance the level of happiness or enjoyment of the user, whereas those might be the most relevant component for the designer of a video game. It is thus important to allow the SETAs to be flexible to the system under test and the goals of the designers.
The same modular concept applies to the behaviour of the SETAs. Different systems under test will require agents that behave differently to test their UX. Even the same system might require

agents that behave in different ways for different scenarios and UX components being tested. For some systems, it might be feasible and logical to test UX with an agent that tries to cover all possible actions and behaviours. For other systems, like a real-life simulation or a complex video game, that might be unfeasible given the action space, as well as irrelevant for UX testing. In most cases, agents that behave randomly would provide unreliable and possibly useless information regarding the UX of a system. A further discussion about the relevant characteristics for the behaviour of UX testing agents will be presented in Section 4.

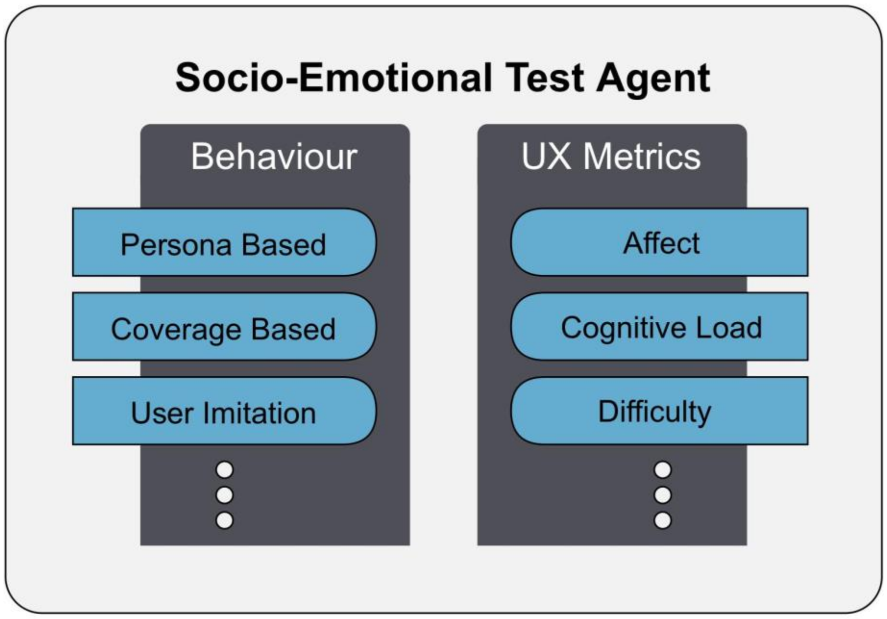


Figure 2: The modular architecture of the SETAs.

We therefore designed our SETAs with a bipartite nature, having a behavioural component, responsible for deciding the actions taken by the agent, and a UX metrics component, responsible for making predictions related to the several components of UX. Both these components are interdependent and needed for a UX testing agent. The behaviour of the agent will affect the measurements done by the UX metrics components, as different traversals of a level or situation will likely lead to different UX.

With this modular SETA architecture (Figure 2), we allow designers and testers to choose which existing modules are most relevant for their needs and even develop new modules which can be easily integrated with the existing architecture and work alongside other modules.

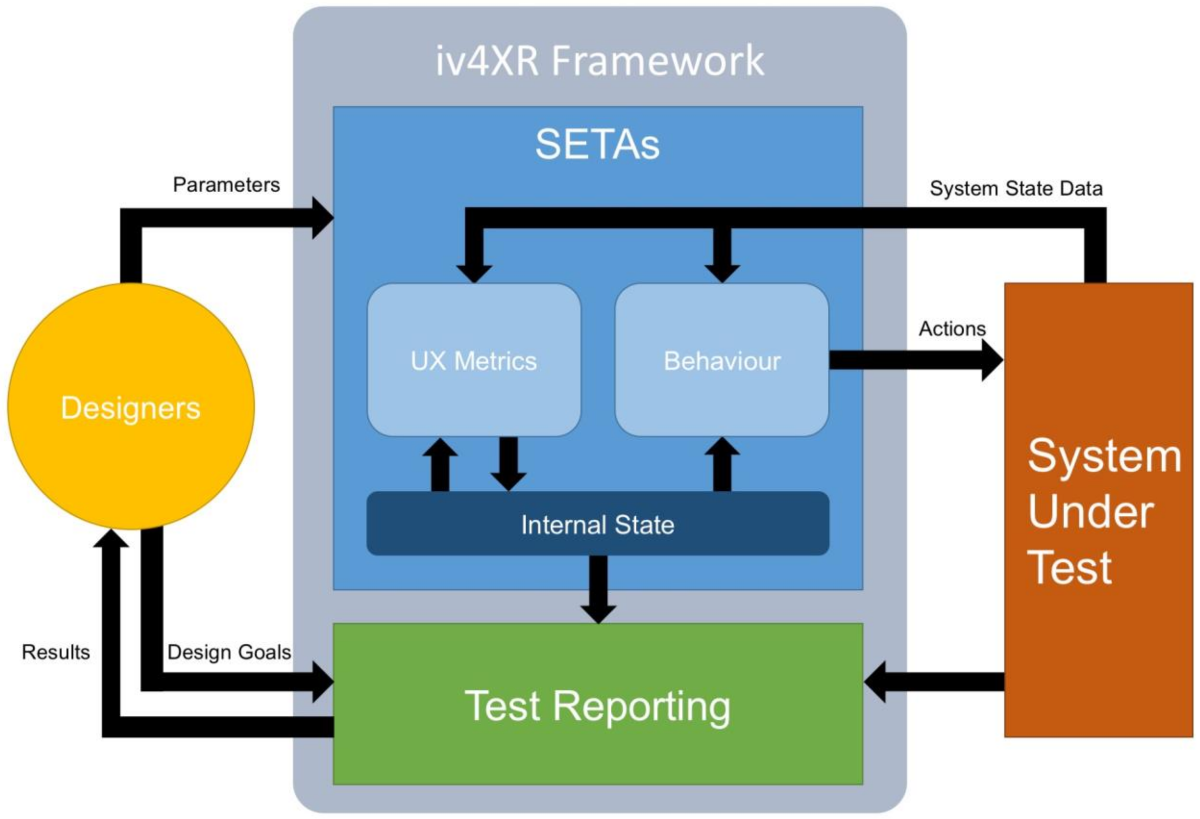


Figure 3: The interaction between the designers, the SETAs and the system under test.

In Figure 3, we find a diagram of the relationship between the designers, the SETAs and the system under test. The designers need to define which UX Metrics modules and Behavioural modules will be used as well as specify the parameters for those modules. By doing so, they define the SETA that best matches their testing needs. Once defined, the SETA will interact with the System Under Test (SUT). The evolution of the state of the system and the internal state of the agent (UX Metrics) will then be used to provide the designer with information about the SUT. If Design Goals have been defined, representing the UX objectives the designer had in mind when designing a scenario, then the designer will also be informed about which scenarios satisfy the Design Goals for the used SETAs and which do not.
In the next section, we will describe the UX metrics modules developed throughout the life-span of the project. We will then describe the behavioural modules in Section 4. We will then give examples of how the combination of behaviour and UX metrics can be used to help designers meet their testing needs in Section 5.

SECTION 3 - UX METRICS

As was discussed in Section 2, UX is a complex concept which has many underlying components. Our approach to testing UX is therefore not to try and measure UX as a whole but to look at these different components and try to test them individually. As such, in the following subsections, we will present the solutions we have developed for the testing of a number of UX components. These modules of UX testing can be used simultaneously or in any combination, depending on the goals of the designer and the system under test.
We will here describe modules for: emotion prediction; cognitive load prediction; motion sickness prediction; difficulty estimation; and validating the plot of interactive narrative games.

SECTION 3.1 - EMOTION PREDICTION

We have developed two modules for emotion prediction, one being model based and the other based on machine learning. The approaches require different types of setup from the designers and testers and are best suited for different systems under test and testing scenarios. The model based approach doesn't require training data but requires expert knowledge. The machine learning approach doesn't require expert knowledge but requires training data. Both approaches will be described in further detail in the following subsections.

SECTION 3.1.1 - OCC PREDICTION - MODEL BASED

User eXperience and emotions are tied to human cognition. Cognitive scientists and psychologists have been investigating emotions and its relation with experience for decades, leading to the creation of theoretical models of emotion that provides a more coherent outlook of cognitive processes. According to appraisal theories of emotions, common patterns can be found in the emergence of the same emotion. These patterns are given as a structure of emotions by the Ortony-Clore-Collins (OCC)[1] theory of emotions. Thus, a model-driven approach derived from a well-grounded theory of emotions, such as OCC, is sensible when access to sufficient data is not possible. Each of the emotion types listed in Figure 4 is specified as described in the OCC.

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

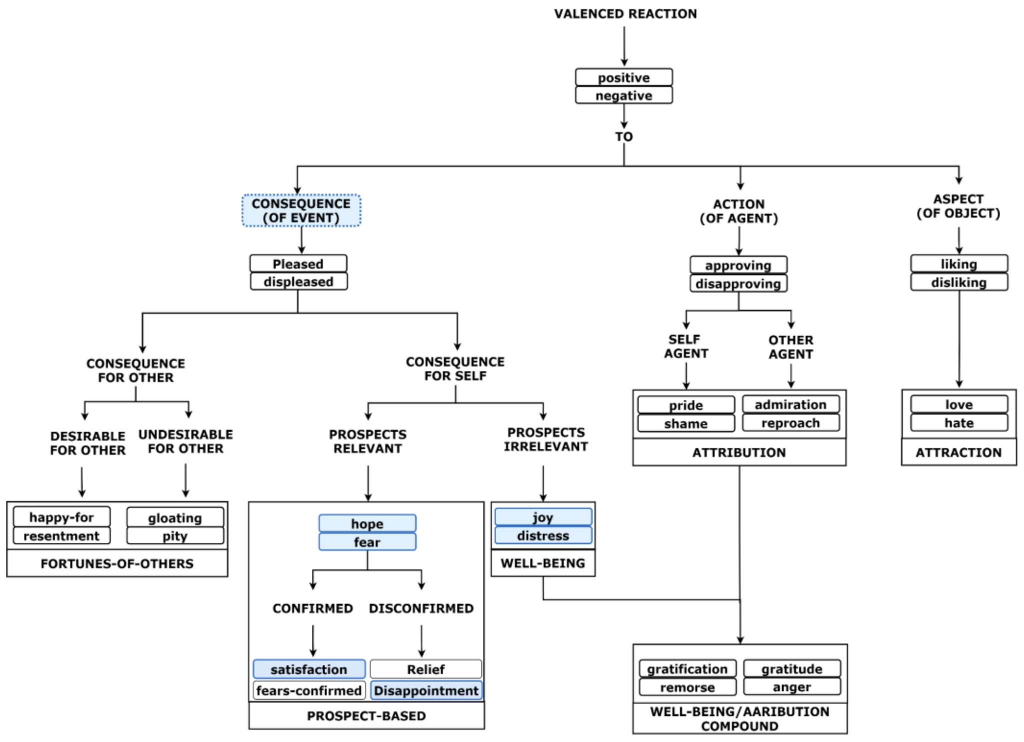


Figure 4: Structure of emotions according to the OCC model

Our desire to develop a model-driven approach led us to introduce a transition system to model event-driven emotions based on the OOC cognitive theory of emotions. It is essentially a *computational model* that can simulate players emotions during a game-play by replacing a human player with an automated agent with an emotion model. The original OCC theory gives a cognitive structure for 22 emotion types, viewed as cognitive processes where each emotion type is elicited under certain conditions (Figure 4).
We introduced a formal model for event-driven OCC emotions using an event-based transition system to serve as the foundation of our automated UX testing approach. These emotions are: hope, joy, satisfaction, fear, distress, and disappointment (six from the aforementioned OCC's full set of 22) for a single agent simulation where the agent's emotional state changes only by system under test dynamism expressed through events to the agent. A given system under test is treated as an environment that discretely produces events triggered by the agent's actions or environmental dynamism such as hazards. The event tick represents the passage of time. The emotion model of an agent is defined as a 7-tuple transition system *M*:

$$(\Sigma, s0, G, E, \delta, Des, Thres)$$

- G is a set of goals that the agent wants to achieve; each is a pair $< id, x >$ of a unique goal name (id) and significance degree (x).
- Σ is the set of *M*'s possible states; each is a pair $< K, Emo >$:
    - K is a set of propositions the agent believes to be true. It includes, for each goal *g*, a proposition status(*g*,*p*) indicating its confirmation or disconfirmation status with *p*

∈ {achieved, failed, proceeding}, if *g* has been achieved or failed, and a proposition $P(g, v)$, with v ∈ [0..1], stating the agent's current belief on the likelihood of reaching this goal.
  - *Emo* is the agent's emotional state represented by a set of active emotions, each is a tuple <*ety, w , g, t₀*>, *ety* is the emotion type, *w* is the intensity of the emotion respecting a goal *g*, and triggered time t_0.
- $s_0 \in \Sigma$ is the agent's initial state.
- *E* specifies the types of events the agent can experience
- δ is M 's state transition function; to be elaborated later.
- *Des* is an appraisal function; *Des*(*K*, *e*, *g*) expresses the desirability, as a numeric value, of an event *e* with respect to achieving a goal *g*, judged when the agent believes *K*. OCC theory has more appraisal factors, but only desirability matters for aforementioned types of emotion.
- *Thres*: thresholds for activating every emotion type.

One or multiple emotions can be activated by an incoming event (except *tick*). This is formulated as follows:

$$newEmo(K, e, G) = \{\langle ety, g, w, t\rangle \mid ety \in Etype, g \in G, w = \varepsilon_{ety}(K, e, g) > 0$$

where *w* is the intensity of the emotion *ety* towards the goal *g* ∈ *G* and *t* is the current system time. Upon an incoming event, the above function is called to check the occurrence of new emotions as well as re-stimulation of existing emotions in Emo for every *g* ∈ G. $\varepsilon_{ety}(K, e, g)$ internally calculates an activation potential value and compares it to a threshold $Thres_{ety}$; a new emotion is only triggered if the activation potential value exceeds the threshold. These thresholds might vary according to players' characters and their moods. For instance, when a person is in a good mood, their threshold for activating negative emotions goes up, which conveys how they become more tolerant to feeling negative-valenced emotions. As examples, the activation functions for hope and fear emotions, based on provided definitions in the OCC theory, are as follows:

- **Hope** is defined as being pleased about the prospect of a desirable consequence of event in OCC theory which is formalised in the defined transition system *M* as follows:
  $\varepsilon_{hope}(K, e, g) = v' * x - Thres_{hope}$
  , provided $g = <id, x> \in G, P(g, v) \in K, P(g, v') \in e(K), and\ v < v' < 1.$

- **Fear** is defined as being displeased about the prospect of an undesirable consequence of event in OCC theory which can be formalised in the transition system *M* as follows:
  $\varepsilon_{fear}(K, e, g) = (1 - v') * x - Thres_{fear}$
  , provided $g = <id, x> \in G, P(g, v) \in K, P(g, v') \in e(K), and\ 0 < v' < v.$

For further explanation of the model and formalisations of emotions, please see[2].

[2] Ansari SG, Prasetya IS, Dastani M, Dignum F, Keller G. An Appraisal Transition System for Event-Driven Emotions in Agent-Based Player Experience Testing. In International Workshop on Engineering Multi-Agent Systems 2021 May 3 (pp. 156-174). Springer, Cham.

The OCC model exists as a UX metrics module of SETAs to address affects and emotions. There are also characteristics that the designer needs to specify in the agent to resemble a certain character of players in the OCC model, such as: a player's goals and the goals priorities, the player's initial mood and beliefs before playing the game, the desirability/undesirability of incoming events for the player and how they affect goals' likelihoods. E.g., a player might experience a high level of positive emotions on defeating an enemy in a game, while for another player who prefers to avoid conflicts acquiring a gold bar could be more desirable. These characteristics are addressed as input parameters of SETAs in Figure 3. PX is the terminology applied to UX in games. Specifically, the model is used for Lab Recruits which is a 3D Unity game which has different replayable levels and is connected to the iv4XR framework.

Figure 6a shows a floor plan of a Lab Recruits level exposed to PX testing using our approach. 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’. Figure 5b and 5c show two provided setups with the different amounts 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. As mentioned earlier in this section, 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 SETAs input parameters. A test agent is deployed, through the iv4XR framework, 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. As the agent progresses, its belief on the likelihood of completing the level changes, depending on the number of opened doors as well as remaining 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.

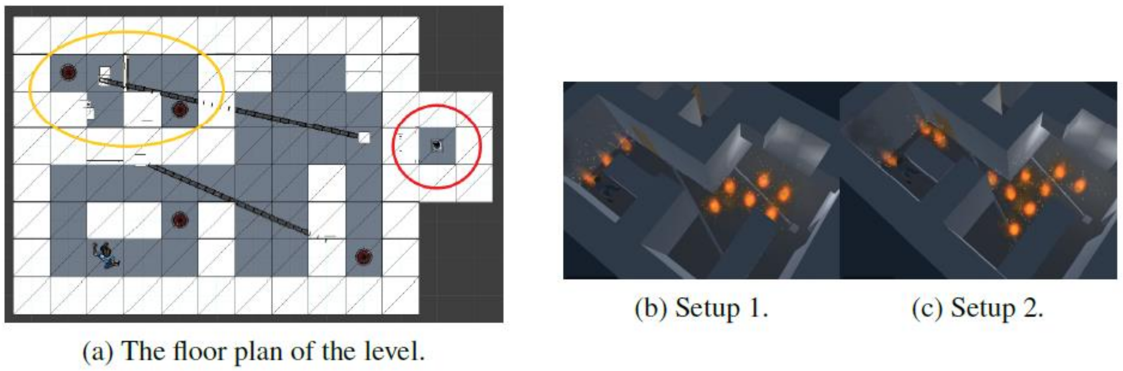
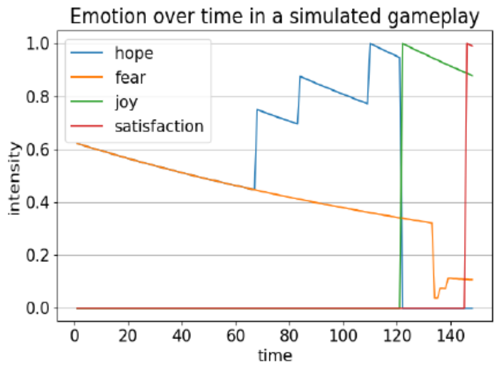


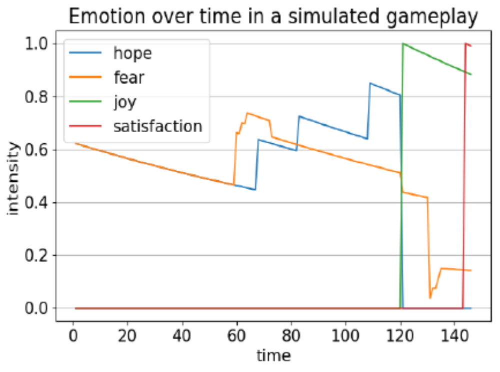
Figure 5: A simple level under test in Lab Recruits with two different setups

With assigning the level under test to SETAs, an intelligent test agent is run through the level using tactics provided in iv4xr framework for the agent to do decision making and behave. Then, the OCC model in UX metrics component decides about its internal emotional states of the agent based on perceived consequences of actions the agent took as well as environment dynamisms. The agent’s emotional state keeps getting updated until the agent successfully reaches the exist

room or dies. Then a report of experienced emotions in terms of timeline graphs and heatmaps will be produced for the designer.
The timeline of emotions in setup 1 and setup 2 in Figure 6 shows that the trend of positive emotions is actually quite similar, although with a smaller level of hope in the setup 2. However, comparing the result of setup 2 with setup 1 reveals something interesting. Fear shows a quite different trend in setup 2 (Figure 6b). It is stimulated multiple times during the execution, whereas the same emotion, despite having numbers of fire hazards, has been never stimulated in setup 1 (Figure 6a). In other words, having some fire hazards may not necessarily trigger fears in the agent unless the agent passes a certain number of fire hazards. Such a comparison can be useful for designers e.g., to determine the amount and placement of hazards to induce a 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 and even in a relatively close time interval with a rise in hope, while still keeping the level survivable. Moreover, this information can be shown in terms of heat maps, providing spatial information of the agent's emotions in a set up. Figure 7 shows heatmaps of positive and negative emotions for setup 2.

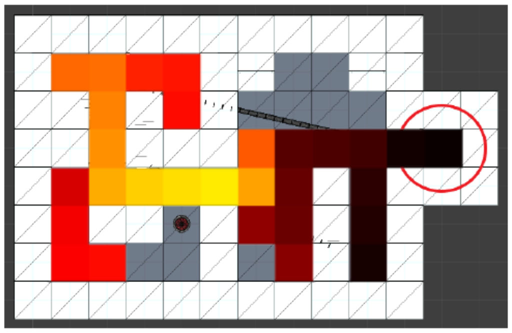


(a) Setup 1

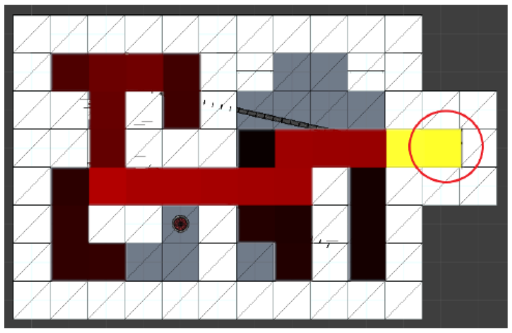


(b) Setup 2

Figure 6: Timeline of emotions in Setup 1 and Setup 2.



(a) Negative emotions: yellow= high fear, dark red=low fear.



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

Figure 7: The heat maps of triggered emotions in setup 2.

Black= very low intensity (or no emotion), white= walls, grey=unexplored area.

The proposed emotional modelling can be used for any designed Lab Recruits level. Figures 8 and 9 show the result of an experiment over a medium size level in Lab Recruits.

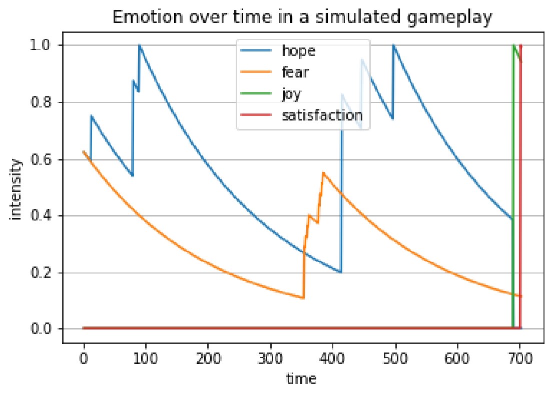
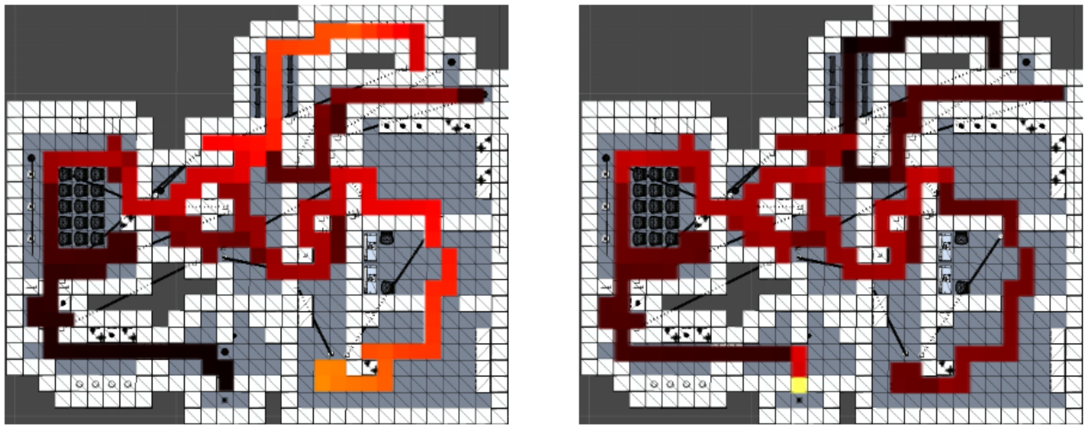


Figure 8: Timeline of emotions in a medium size level called Lab1 in Labrecruits



(a) Negative emotions: yellow= high fear, dark red=low fear.

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

Figure 9: Heat maps of emotions in Lab1

SECTION 3.1.2 - EMOTIONAL PREDICTION USING THE PAD MODEL AND MACHINE LEARNING

Another of the implemented modules of our SETAs that focuses on emotional prediction is based on the PAD model of emotion[3]. This model describes human emotions based on three dimensions: Pleasure; Arousal; and Dominance. For this module, we used machine learning to train a predictive model for the dimensions of the PAD model based on data collected from the system under test. A representation of the machine learning process can be found on Figure 10.

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

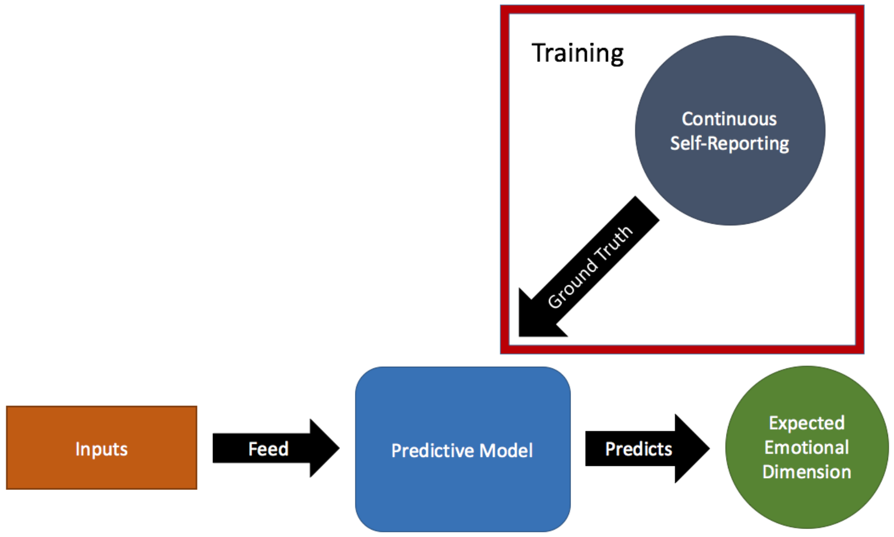


Figure 10: A graphical representation of the machine learning approach. The main idea is to use as training inputs the internal state of the system under test along with the state of the agent. Using these inputs, we can then train a predictive model to predict the emotional dimensions based on continuous self reporting of users. The use of user self reporting might appear to go against the objective of automation, but the user traces would only need to be collected once to train the model. Once the model is trained, it can then be used whenever necessary without the need of further user data.

For the sake of training the model, a game named “Flower Hunter” was developed, inspired by old-school top-down 2D games like Legend of Zelda. It was designed to be 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 Figure 11.

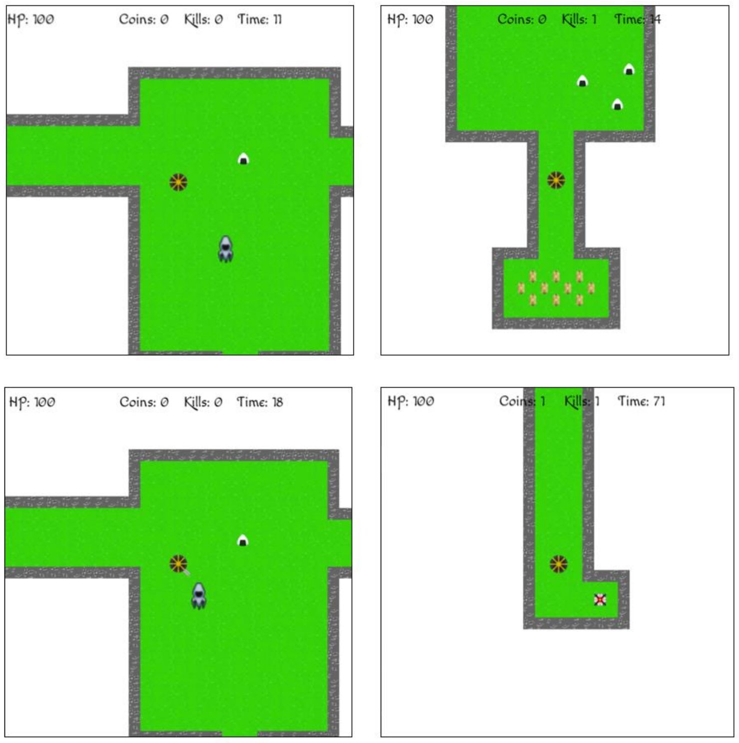


Figure 11: Screenshots of the "Flower Hunter" game. The player, represented by a black and yellow ball, needs to traverse a field riddled with enemies, health providing rice cakes, and coins, in order to find a pink flower (shown on the bottom right screenshot). Once the player finds this flower, the game is over and the player won. If the player loses all its health points by touching enemies, then the game ends and the player lost. The player can use a sword to fight and kill the enemies (shown in the bottom left screenshot).

The inputs used to train the predictive model were all numerical in nature and encoded the time passed since several relevant events in the game, the number of each type of agent seen by the player and the distance to those objects. These inputs were chosen to be as abstract as possible and thus applicable to many different genres of games and simulations.
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. As such, we decided to use continuous, after the fact, annotation, inspired by previous works that used a similar approach[4,5].

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

[5] Melhart, D., Liapis, A. and Yannakakis, G.N., 2021. The Affect Game AnnotatIoN (AGAIN) dataset. arXiv preprint arXiv:2104.02643.

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. He 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 Figure 12.

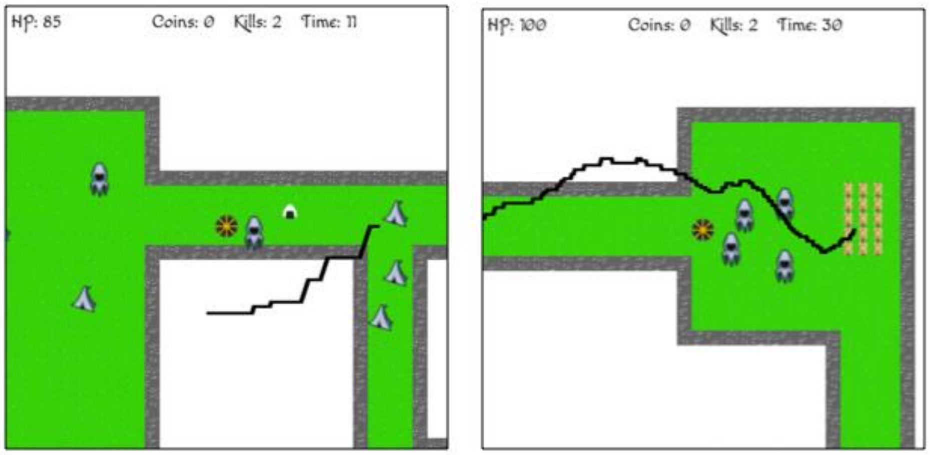


Figure 12: Screenshots of the continuous annotation process used. The black line that is seen on the screen was controlled by the user, going up or down according to the perception the user had of the dimension being annotated having increased or decreased, respectively.

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 he 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 had 91 participants play three "Flower Hunter" levels and self report 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. These participants were students of Psychology, which might enable them to more easily understand the definitions of the PAD dimension. The levels used for the experiment can be found on Figure 13. These maps had the same topology but different distribution and quantity of objects and enemies. In the end, we had 264 annotated traces.

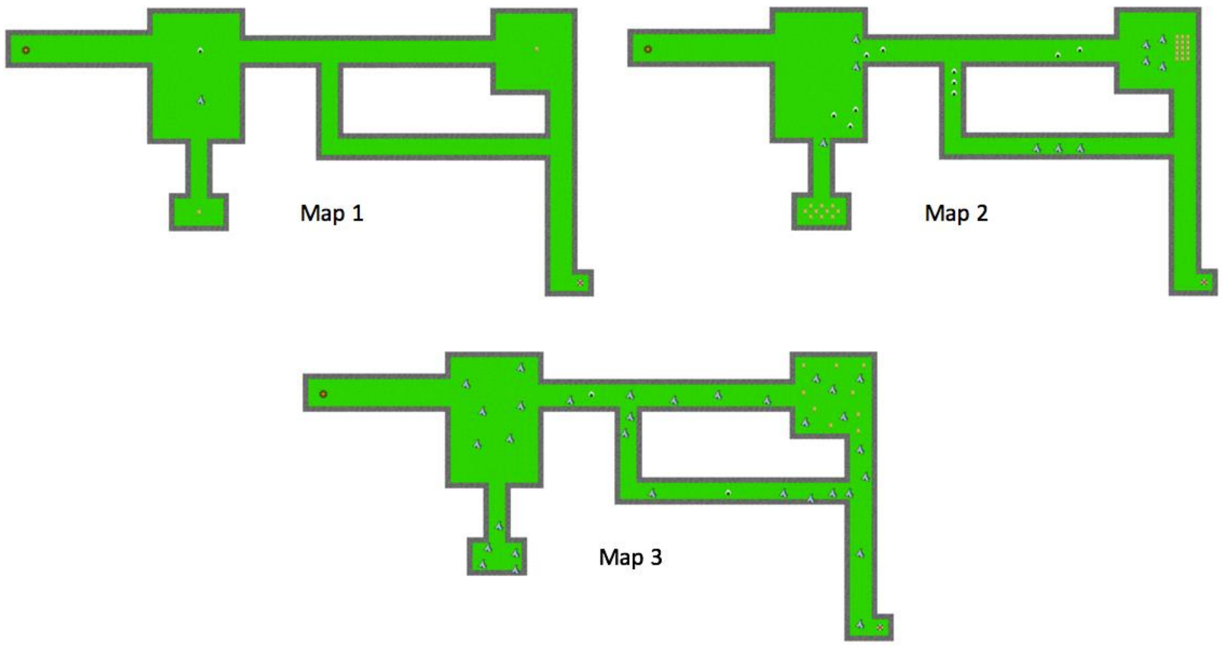


Figure 13: The three maps used for the collection of data. As can be seen, all the maps had the same topology yet the location and number of objects differed between them.

The several input values and the output value were collected with a frequency of 8 Hz. 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 3 traces where there was no change to the emotional dimension throughout the entire play-through. 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.

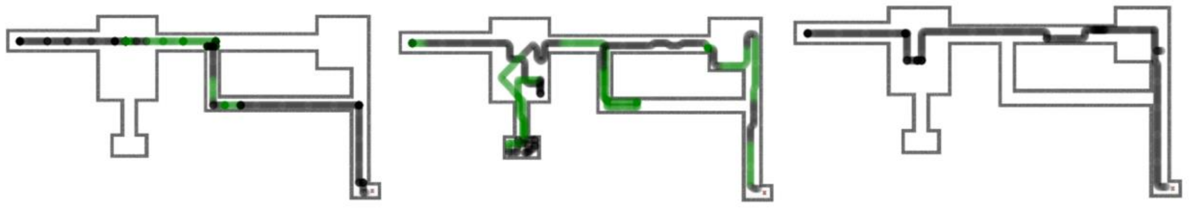


Figure 14: Three examples of traces and self reporting for the Arousal dimension. These traces are for Map 2 and show how different users can have very different experiences depending on the traversal of the map and their own different perceptions for the reported dimension.

After gathering and processing the data as described in the previous section, 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. The accuracy was based on the correct classification of 1 second slices using the "leave one out" method. 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 ≈60% accuracy.

SECTION 3.2 - COGNITIVE LOAD PREDICTION

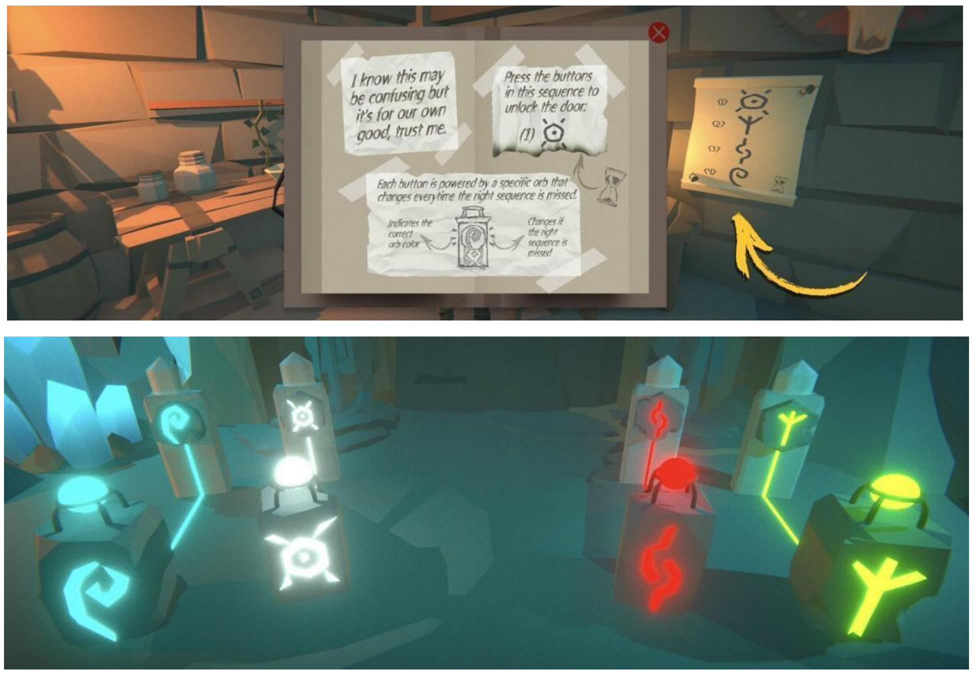
A significant part of XR user experience results from the interaction of each user with the system while solving a task in the environment. During interaction with an XR system, each user has different exchanges while navigating through the design space of the system, which will create distinct user 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.
We addressed the prediction of 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. To create this toolset, we explored two different approaches. For the first one we focused on the TBRS (Time-Based Resource Sharing) memory model. For our second approach we created a plugin that follows a secondary task paradigm. The toolset,

based on the TBRS memory model[6], 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 computes an estimate for the cognitive load experienced by the user, based on the sum of the attention-grabbing events, i.e., the total amount of time in which a participant was fully paying attention to a task, and the total duration of the task.

$$CL = \frac{\sum_{i=1}^{N} a_i}{T}$$

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 15), 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.
We also worked on a plugin that allows for a more fine-grained measure of cognitive load. The plugin introduces a secondary task in the game. Secondary task paradigm, common in memory studies, introduces a secondary task that competes for the same resources as the main task.

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



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 15: Screen shots from the 'Way out' game

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 started 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 created two versions of the secondary task. One is integrated in the game scenario and the other is not. For the integrated version we tell participants that sometimes the golem (the main character) overheats from time to time. When they see smoke coming out of the golem's head, the participant needs to press the spacebar to prevent the golem from overheating. If they let the golem overheat three times, the game finishes. On the other version (not integrated in the game narrative), the participants are told that a red dot will appear from time to time in the centre of the screen, when that happens, they need to press the spacebar. If they miss three times they lose the game. We are currently in the process of collecting data for this experiment. We hope to have a tool that developers can use to assess the cognitive load imposed by their games (the plugin) and also want to check the data being collected against the attention grabbing events identified in the first study to see if they remain a good predictor of Cognitive Load.

SECTION 3.3 - MOTION SICKNESS PREDICTION

Cybersickness (CS) is one of the factors that can significantly hinder user experience in immersive Virtual Reality (VR) games and simulations. CS is a malady that may cause feelings of Motion

Sickness (MS) such as nausea, general unwellness, blurry vision, among others, as a result of VR immersion. While there are techniques to reduce the likelihood of CS occurrence, developers cannot confidently foresee their need as current CS severity measuring tools, such as questionnaires, detect it only post-factum, thereby requiring intensive human testing to assess what CS reduction measures must be taken.

With this in mind, we have developed a machine learning model capable of automatically predicting CS occurrences of a VR experience based on Head-Mounted Display (HMD) eye-screen video and user control input. We collected data from 22 participants while immersed in a stylized flight simulator VR game depicted on Figure 16 that we used to develop the model. Participants reported feelings of CS, at every minute, using the Fast Motion Sickness Scale[7]. We applied several machine learning approaches and finally settled in a Random Forest model that is able to predict CS with an accuracy of 72.9%. The algorithm used features like the colour of the image pixels, the video optical flow, and user input to predict the occurrence of CS in two classes (present or not) for each minute of video. Our approach is more suited to automated testing than the previous approaches, as it only needs the actions performed and the video presented to the user, while others[8] depend on physiological measures of players.

We have yet to test whether the predictive model trained using one VR game would be applicable to another game with similar levels of accuracy or if new training is required for each different game. Even if new training is indeed required, the approach can nonetheless be used to save countless hours of user testing, as the predictive model can be reused as many times as needed after being trained.

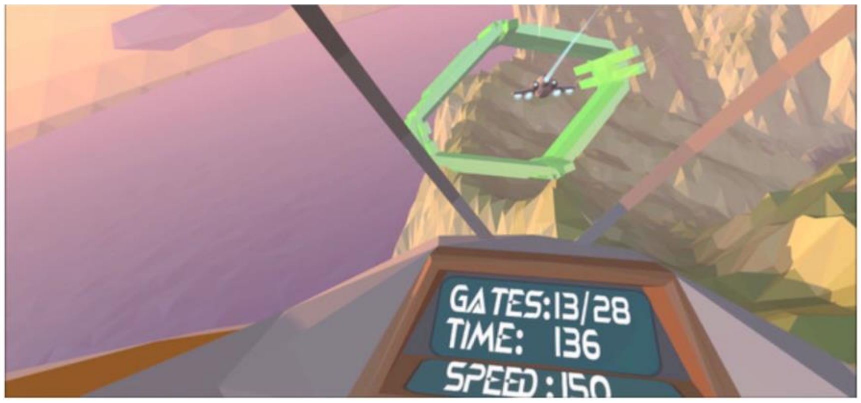


Figure 16: Faceted Flight - A stylized stunt flight VR simulator that was used to develop and test our CS predictive model.

[7] B. Keshavarz and H. Hecht. Validating an efficient method to quantify motion sickness. Human factors, 53(4):415–426, 2011.

[8] R. Islam, Y. Lee, M. Jaloli, I. Muhammad, D. Zhu, P. Rad, Y. Huang, and J. Quarles. Automatic detection and prediction of cybersickness severity using deep neural networks from user's physiological signals. In International Symposium on Mixed and Augmented Reality (ISMAR), pp. 400–411. IEEE, 2020.

SECTION 3.4 - 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 have developed 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 ranking is done by training agents to solve the levels based on reinforcement learning and then adding noise to the solution. The lesser the amount of a given type of noise needed to make the agent unable to solve the level, then the greater the predicted difficulty of the level. An example ranking by increasing difficulty of the levels of a platformer like game based on timing errors can be found in Figure 17.

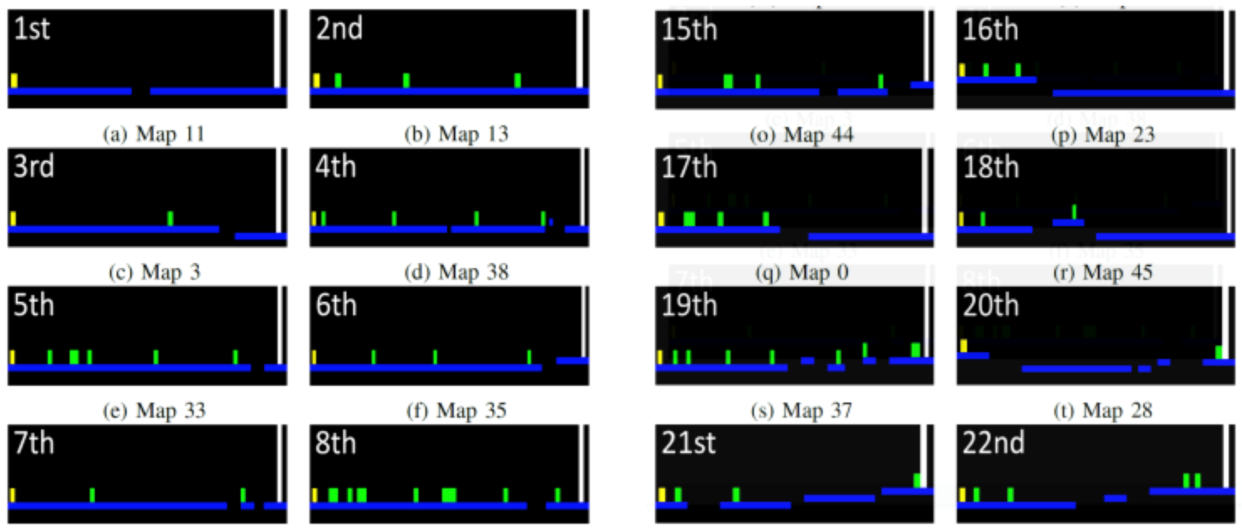


Figure 17: Ranking of a set of 22 levels based on increasing difficulty.

The indicators of difficulty also help evaluate the hardness of finding a weakness in the system under test. In the intrusion scenario for testing the patrol of a powerplant in the MAEV platform, one of the pilots of the iv4XR project, we use indicators of difficulty in order to evaluate how good the patrol is. The indicators are based on analysing the interaction of the intruder agent with the patrol. Moreover, in the case of the powerplant use-case, we not only have one intruder agent but a full archive composed of a population of successful RL intruder agents. This is the case because, to solve the powerplant problem, we used a QDRL (Quality-Diversity Reinforcement Learning) approach that tries to find multiple and diverse solutions to the same RL problem. We leverage the population of RL intruders in order to compute more precise indicators. We used three types of indicators. The first one is the stress indicator, being the average distance during one intrusion episode between the intruder RL agents and the fixed patrol guards. Figure 18 below is a visualisation example of how each intruder from the QDRL archive is stressed by the proximity of the guards (in blue) as illustrated by the colour of each intruder going from green (not stressed) to red (stressed).

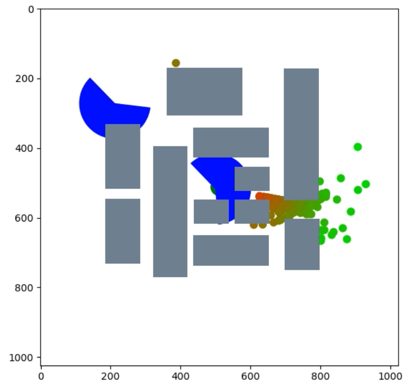


Figure 18: the stress indicators on how close the intruder is to the patrol.

The second indicator is the delay indicator. For each agent we compute how sensitive they are to delays in the execution of their actions. If adding delays to execution of the actions is harmless to the performance of the agent, then it is an indicator that the difficulty is small. We report in Figure 19 (left) how our agents' performance evolve with delays. Each curve corresponds to a different intruding scenario with a different patrol pattern. The y axis is the percentage of agents in the population that successfully intrude, and the x axis is the amount of the delay added to the execution of an action.

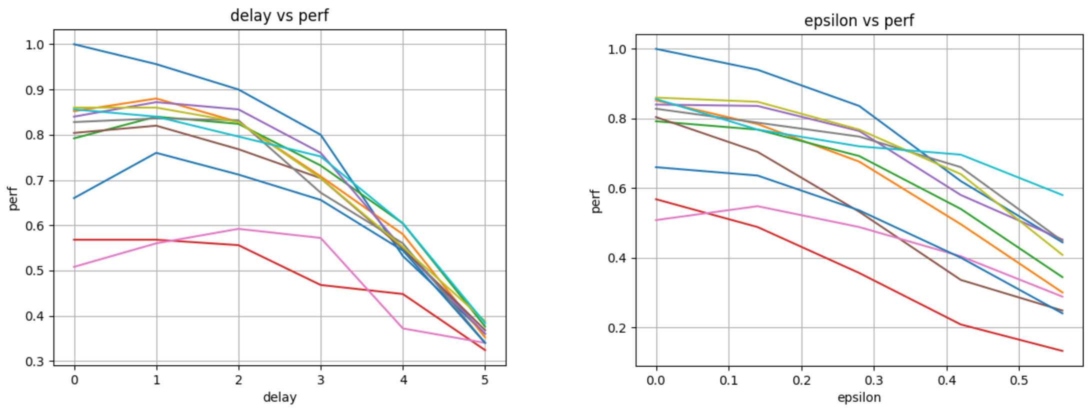
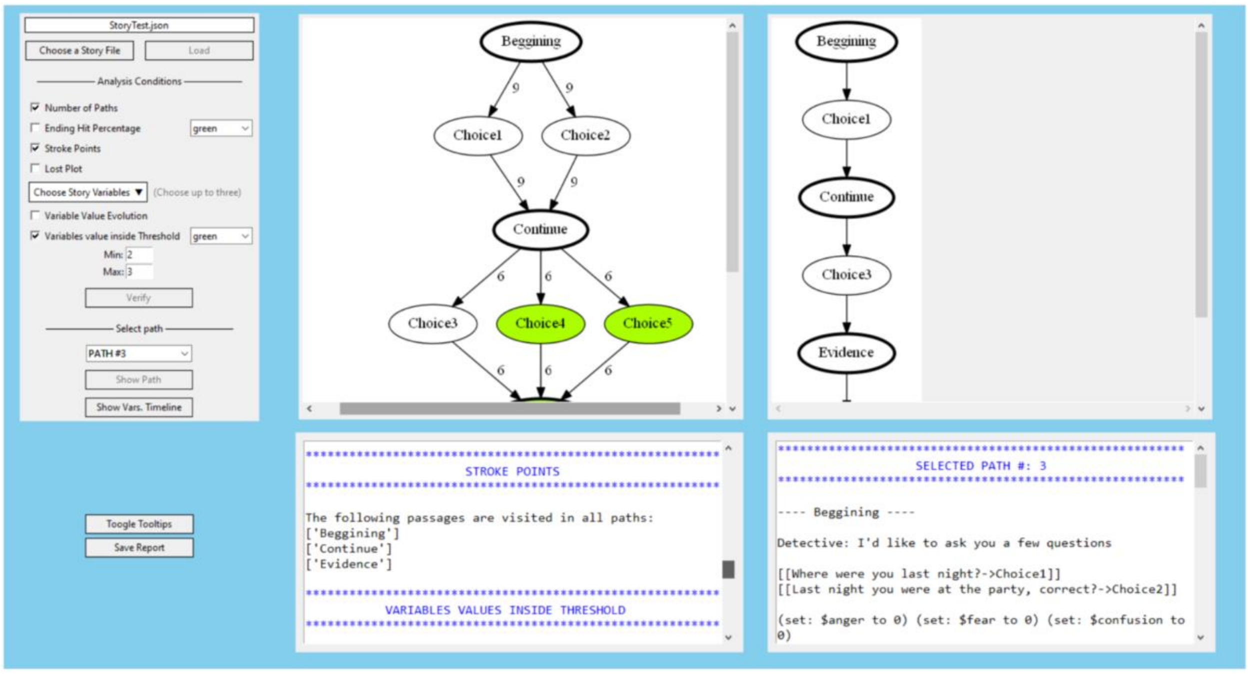


Figure 19: Delay indicator, robustness to delay in the execution of the action.

The third indicator is the noise indicator. For each agent we compute how sensitive they are to noise added to their actions. If adding large noise of the actions is harmless to the performance of the agent, then it is an indicator that the difficulty is small. We report in Figure 19 (right) how our agents' performance evolves with noise. Each curve corresponds to a different intruding scenario with a different patrol pattern. The y axis is the percentage of agents in the population that successfully intrude, and the x axis is the amount of the noise added to an action.

SECTION 3.5 - 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 study 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 (Figure 20), 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. Full details of the model and evaluation can be found in the published paper[9].



☑ Number of Paths
☐ Ending Hit Percentage: green
☑ Stroke Points
☐ Lost Plot
☐ Variable Value Evolution
☑ Variables value inside Threshold: green
Min: 2
Max: 3
PATH #3

```
*************************************************************
                       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)
```

Figure 20: The Story Validator GUI.

[9] Carolina Veloso and Rui Prada: "Validating the plot of Interactive Narrative games" in proceedings of CoG 2021- International Conference on Games, pp. 1-8, Copenhagen, Denmark, August 2021. IEEE.

SECTION 4 - UX BEHAVIOUR

The behaviour and choices of a user when using a complex system will heavily impact the UX of that user. Similarly, the behaviour of a UX testing agent will also impact the UX predictions made. As such, it is important to ensure the behaviour of UX testing agents is adequate and representative of the users that might use a system. We have thus also conducted research on how to achieve this, which will be presented in the following subsections.
In other Work Packages of the project, other behavioural modules were developed, focusing more on functional testing than UX testing. These behavioural modules can, nonetheless, be used with the previously described UX modules for the testing of UX much like the UX based behavioural modules would be used. The accuracy of the UX predictions could suffer from using behaviour that is not tailored for UX testing, but the information provided could still be valuable for developers and testers. Having UX information when running functional tests could also be used in order to save testing time in systems under test where running simulations with agents takes a significant amount of time.

SECTION 4.1 - COVERAGE BASED BEHAVIOUR

As mentioned in 4.2, PX testing is aimed to assist game designers to test the expected emotional experience during a game play as patterns of emotions that need to be verified over time and space. If these patterns do not fulfil design intentions, game properties can be altered, and the testing process can be repeated. To meet this aim, it is essential to have a set/suite of game executions that cover different player behaviour to assure that the UX testing results are reliable. Figure 21 shows the general architecture of PX testing, using as foundation the OCC predictive model of emotion explained in Section 4.2. It has four main components: a Model-based Testing component for generating tests, the Model of Emotions component implementing the aforementioned computational model of OCC emotions, a basic test agent for controlling the in-game player-character, and the PX Testing Tool as an interface for a game designer towards the framework. The designer needs to provide the following inputs, which can also be seen in Figure 21 (1):

1. An extended finite state machine (EFSM) that abstractly models the functional behaviour of the game under test.
2. A selection of which game events have impacts on the player’s emotions (e.g. defeating an enemy, acquiring gold).
3. Characteristics that the designer wants to address in the agent to resemble a certain type of players, such as: a player’s goals and their priorities, the player’s initial mood and beliefs before playing the game, and the desirability of incoming events for the player. E.g. a player might experience a high level of positive emotions on defeating an enemy, while for another player who prefers to avoid conflicts, acquiring a gold bar could be more desirable.

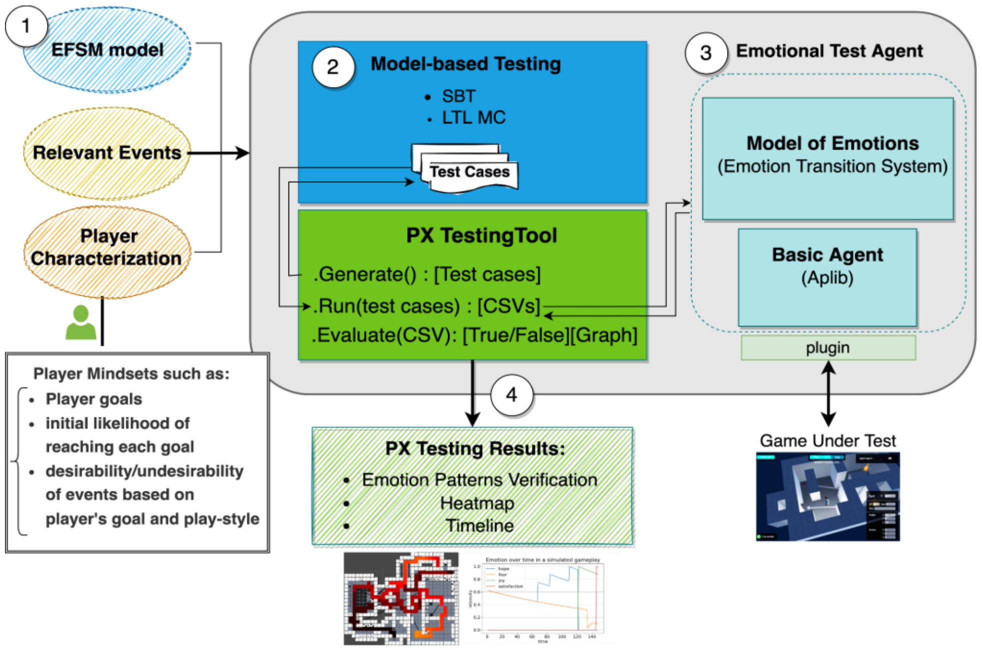

1
EFSM model
Relevant Events
Player Characterization
Player Mindsets such as:
• Player goals
• initial likelihood of reaching each goal
• desirability/undesirability of events based on player's goal and play-style
2
Model-based Testing
• SBT
• LTL MC
Test Cases
PX TestingTool
.Generate() : [Test cases]
.Run(test cases) : [CSVs]
.Evaluate(CSV): [True/False][Graph]
3
Emotional Test Agent
Model of Emotions
(Emotion Transition System)
Basic Agent
(Aplib)
plugin
4
PX Testing Results:
• Emotion Patterns Verification
• Heatmap
• Timeline
Game Under Test

Figure 21: Architecture of PX testing foundation using OCC-emotion model.

Given the EFSM model, the Model-based testing component, (2) in Figure 21, generates a test suite consisting of abstract test cases to be executed on the game under test (GUT). To generate a test suite with a model-based approach, the EvoMBT[10] tool developed for the iv4XR framework is used. Due to the abstraction of the EFSM model in EvoMBT, emotion traces cannot be obtained from pure on-model executions. They require the executions of the test cases on the game under test (GUT). An adapter is needed to convert the abstract test cases into actual instructions for the GUT. The basic test agent does this conversion. Attaching the Model of Emotions to the basic test agent creates an emotional test agent, (3) in Figure 21, which is able to simulate emotions based on incoming events. Via a plugin, the emotional test agent is connected to the GUT. Each test case of the test suite is then given to the agent for execution. The agent computes its emotional state upon observing events and records it in a trace file. Finally, when the whole test suite is executed, the PX Testing Tool component analyses the traces to verify given emotional requirements and provide heat-maps and timeline graphs of emotions for the given level (4) in Figure 21.

Evaluation of emotional heat-maps. A heat-map shows patterns over space. It can be constructed for every test case, but we can also construct an aggregated heat-map of an entire test suite by merging the traces of its test cases into a single trace, and then calculate the map from the combined trace for every emotion as seen in Figure 22.

[10] https://github.com/iv4xr-project/iv4xr-framework/blob/main/docs/mbt/MBT.md

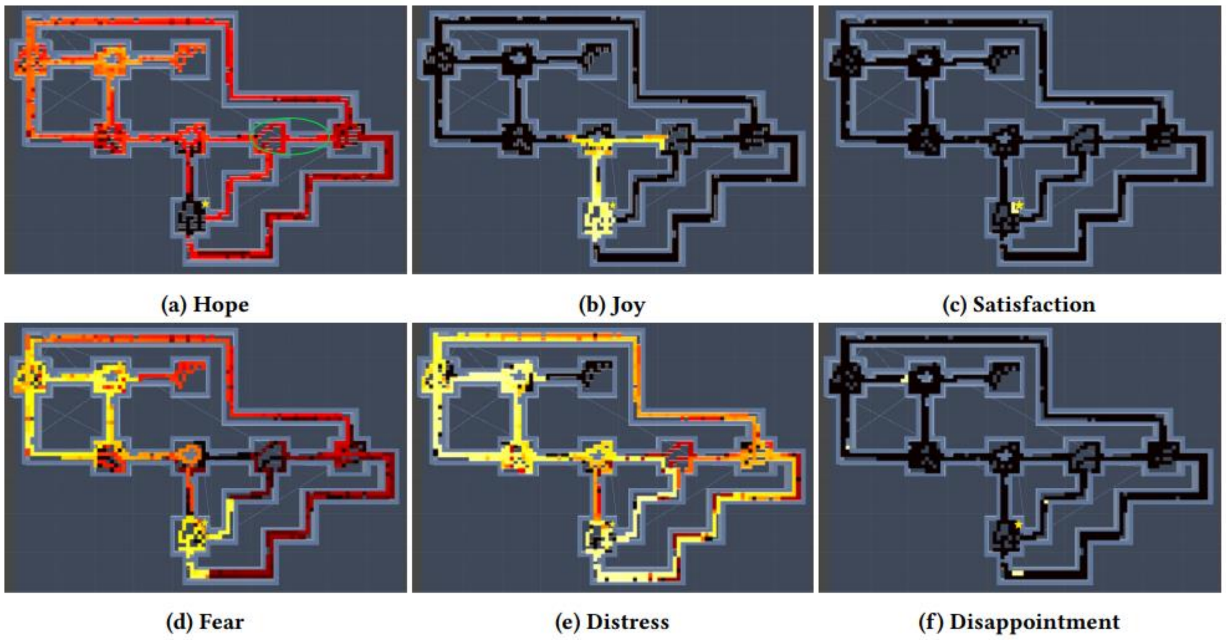


Figure 22: Aggregated Heat maps of emotions for all executed test cases

UX Testing and Goal Definition. The testing of UX raises a non trivial problem: how can a tester define a test and the conditions over which it passes or fails? A desired (or undesired) experience can be a complex and spatio-temporal phenomena which cannot be easily verified by the common method of using “asserts” employed by traditional functional testing. A designer might have a very specific arousal experience that she wants users to have and requires a method of expressing that experience in a way that can be automatically verifiable.
As part of the work done in WP4, we have developed a Linear Temporal Logic (LTL) based language which allows designers to specify their UX requirements over time and space. This language was developed with a focus on the testing of user emotions, but could be expanded to take in consideration other components of UX.
Designers can also go beyond the testing of specific emotional experiences such as fear, hope and joy during a game play as patterns of emotions that need to be verified over time and space. If these patterns do not fulfil design intentions, game properties can be altered, and the testing process can be repeated. Although there are numerous combinations of emotions that can be examined for a game level, only a proportion of them matter for the designer to check.

Formal emotion pattern Specifications for testing. The component is capable of verifying emotional requirements given as formal patterns of emotions. We define emotion patterns to capture the presence or absence of an emotional experience in a game. Such a pattern is expressed by a string of symbols, each representing the stimulation (intensity rise), or lack of a certain emotion type.

Definition. An *emotion pattern* is a sequence of stimulations e or $\neg e$, where e is one of the symbols H, J, S, F, D and P. Each represents the stimulation of, respectively, hope, joy, satisfaction, fear, distress, and disappointment.

For example, the pattern *JFS* is satisfied by traces where the agent at some point becomes satisfied (*S*) after a stimulation of joy (*J*), but in between it also experiences a stimulation of fear at least once. Another example is *J¬F S*. when there is no stimulation of fear between joy and satisfaction. The presence of this pattern indicates the presence of a 'sneak' route, where a goal can be achieved without the player having to fight enough for it. Although there are numerous combinations of emotions that can be examined for a game level, there is only a proportion of them that matters for the designer to check. As a requirement, recall that a pattern can be posed as an existential requirement, i.e. *Sat(p)*, or need to happen for all game-plays, i.e. *Valid(p)* or need to unwitnessed for all game-plays, i.e., *USat(p)*. It is also essential to clarify that the choice of which emotion patterns are to be required can vary among game-levels, as expectations on the occurrences of patterns depend on the design of the levels. E.g. a game level with Sat(*DHS*) would provide at least one thrilling game-play. But if it is intended to be an easy level for beginners, the designer might want to insist on *UnSat(DHS)* instead. We have collected a number of emotion pattern requirements from the designer of the specific level, ranging from some simple ones, to more complex ones. The main expectation of the designer is to ensure that the designed level is enjoyable by experiencing different positive as well as negative emotions during the game-play and to avoid the player getting bored. We interpret boredom as the absence of active emotions in the agent's emotional state for some time. As can be seen in Table 1, while most requirements are verified during the test, there are requirements like *Sat(J¬S)* that are failed. This requirement indicates the designer expects at least one execution path where joy is stimulated at least once thought the execution, but the agent never reaches the goal with satisfaction. Finding *Sat* patterns that fail to be witnessed, or *UnSat* patterns that are witnessed, assists the designer to further change their game level and run the agent through the level again for further testing. For example, here, the fail on *Sat(J¬S)* is an indication the designer needs to put some challenging objects like fire or enemies in the vicinity of the goal, when the agent has the goal in sight.

Table 1: emotion requirement verification using iv4xr model-based generated test suite. H= hope, F = fear, J= joy, D= distress, S= satisfaction, P = disappointment and ¬X = absence of emotion X.

Emotion patterns	TS_{SB}
$Sat(\neg DS)$	✔
$UnSat(\neg FS)$	✔
$Sat(J\neg S)$	✗
$UnSat(JD)$	✔
$Sat(JFS)$	✔
$Sat(DHP)$	✔
$Sat(DHS)$	✔
$Sat(DH\neg DS)$	✔
$Sat(FDHFJ)$	✔
$Sat(HFDDDHFJ)$	✔
$Sat(FDDHFP)$	✗

The full explanation of the extended pattern language to write formal emotion specifications based on linear temporal logic is accessible online[11].

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

SECTION 4.2 - PERSONA AGENTS - HUMAN-LIKE BEHAVIOUR

How does a player play? For most games and simulations, there is not a single answer. In the exact same scenario, two different users can behave in completely distinct ways. This makes the task of simulating user behaviour a complex one. In most games, it is unfeasible to simulate every possible sequence of actions a player might do. Even when this is possible, many of those sequences of actions might be highly unlikely to be chosen by a player, whereas others might be extremely common. Knowing which sequences of actions better reflect the behaviour of real users can help developers and testers make better decisions.

In this section, we describe a pipeline for generating persona agents, that is, agents whose behaviour represents a subset of users. To do so, and having already collected user traces, we will begin by clustering those user traces in order to identify persona clusters, meaning subsets of users that behave similarly. We will then use an evolutionary algorithm to evolve genetic agents that mimic the behaviour of the representatives of the identified persona clusters. Having done so, we will thus obtain a set of persona agents.

We describe here the implementation of the pipeline in the game "Flower Hunter", previously described in Section 4.1. Using 91 player traces, we are able to generate 8 persona agents that successfully represented 89% of the players.

To have agents that imitate a subset of users, we must first define these subsets of users, which will henceforth be called persona clusters. If a game or system already has a defined set of persona clusters, then this step is already accomplished, and the sets can be used as they are. However, this will not be true for most systems. To obtain persona clusters from data, we can run a clustering algorithm on collected behavioural traces from real users. By doing so, we will obtain clusters that contain traces that are similar between themselves. The degree of similarity between traces on a cluster and the relevant information for the clustering will depend on the algorithm used.

Once we have the persona clusters, we will need to design agents that behave in accordance to each one of these clusters: the persona agents. We propose achieving this by using genetic agents, being agents that have their behaviour defined by a set of parameters, along with an evolutionary algorithm. By doing so, defining a persona agent becomes finding the set of parameters that characterises the genetic agent that best represents a given persona cluster.

A diagram of the pipeline for defining persona agents can be found in Figure 23. To use this pipeline on any given system, four things need to be implemented: a behavioural distance metric, a clustering algorithm, a genetic agent, and an evolutionary algorithm. With these four components, user traces can be turned into persona agents. The quality and coverage of these persona agents will obviously depend on the implementation of the components as well as on the quantity and quality of collected user traces.

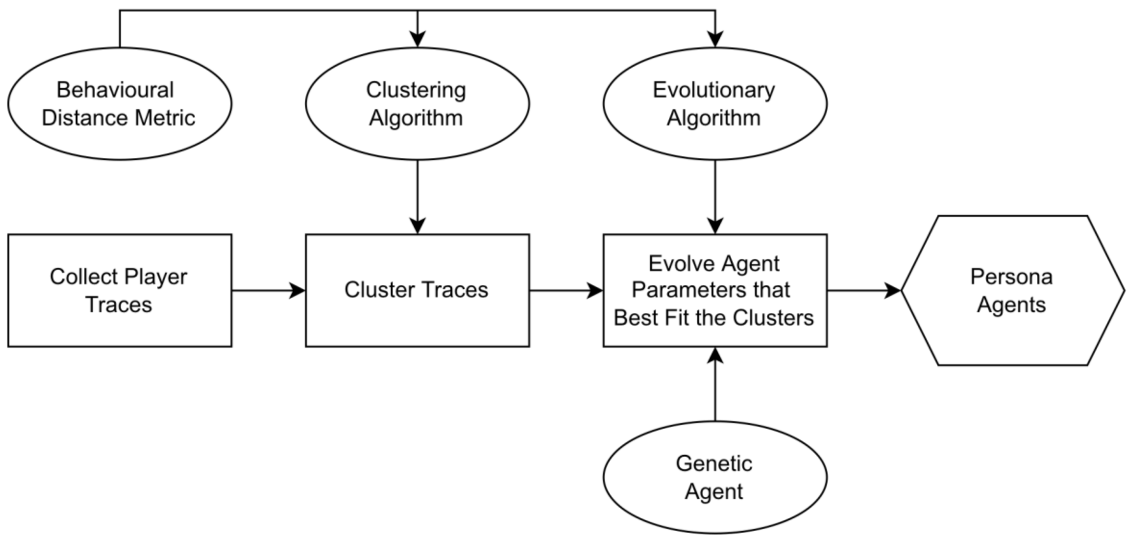


Figure 23: Diagram of the persona agent pipeline, identifying the 4 main components that need to be implemented: a behavioural distance metric; a clustering algorithm; an evolutionary algorithm; and a genetic agent.

We had 91 users play a level of the “Flower Hunter” game (Figure 11), the majority of which were first year students of a Psychology bachelor. These users were explained the mechanics of the game and given the liberty to play however they chose, with no time constraints. Key-pressed and other in-game data were collected, like the position of the player in the map and the positions of the game objects. These form collected traces that serve as the data-set for both training and validating the pipeline here described.

Behavioural Distance Metric. To define a behavioural distance metric, we must decide what constitutes similar behaviour, as the metric will attribute a value of similarity (or dissimilarity) to any given pair of user traces. To define a behavioural distance metric that is the best for every single system and situation is not a trivial task, if at all possible. It might be of interest to a game designer to consider users that choose to go left instead of right as having different behaviours, whereas another designer or even the same designer trying to evaluate a different thing, might prefer to consider them as the same behaviour as long as both users are "exploring".
We decided to consider that players that take different routes and choose different actions when playing a level of the "Flower Hunter" game are behaving differently. Behaviour can then be seen as the sequence of low level actions chosen by a player. We chose to consider the lowest action level, that is, the actions dictated by the keys being pressed at each of the game's time steps. A player game trace can therefore be represented by a string such as "dddwsd", meaning the player pressed the "d" key for the first three time steps, then the key "w" in the fourth, followed by the keys "s" and "d" in the fifth and sixth time steps respectively.

To match this decision, we implemented a distance metric based on the Levenshtein distance[12] between the string representation of the sequence of low level actions. We therefore judged the similarity between two game traces based on the minimum number of edits, deletions and additions that are required to transform the string representation of one of the traces into that of another. For example, a trace represented by the string "dddwsd" has a Levenshtein distance of 4 to the trace represented by the string "wwddsdd".

Clustering Algorithm. Having settled on a metric of behavioural distance, we had now a foundation to choose the clustering algorithm we would be employing to find the persona clusters. We did not have previous knowledge that allowed us to predict the number of persona clusters we could expect to find. Therefore, we had preference for using a clustering algorithm that did not require the number of clusters to be given as a parameter. We also required a clustering algorithms that did not rely on Euclidean distances and could be used with the behavioural distance metric we had previously defined. It would also be a bonus if the clustering algorithm could determine which player trace was the most representative of each persona cluster.
With the aforementioned in mind, we decided to use the affinity propagation clustering algorithm[13], as implemented by the scikit-learn python library. This algorithm allows for any distance metric to be used, defines the number of clusters automatically, and returns the player trace that is most representative of each cluster.

Genetic Agent. A genetic agent is an agent whose behaviour is dictated by the values of a set of parameters, which can be thought of as the genome of the agent. Different sets of parameters will produce different behaviours. This encoding of the agent's behaviour into a set of parameters allows the use of evolutionary algorithms[14] as a way to find genetic agents that behave in a particular way. In this section, we will present our implementation of a genetic agent for the "Flower Hunter" game.
For the agent to represent player behaviour, we had to endow it with the ability to do that which players do. Namely, the agent had to be able to move and explore the map, fight enemies, collect items and reach the final objective. To achieve this, we began by endowing the agent with the ability to navigate the map.
By creating a graph of the visited locations of the map, the agent was able to navigate by finding the shortest path to a given known location and following it. Having this, the agent could also reach any object it found, including enemies. Endowing the agent with the ability of fighting enemies was only a question of having the agent move towards the closest enemy and begin attacking when the enemy was within reach. Our agent was thus able to do everything required

[12] Levenshtein, V.I., 1966, February. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady* (Vol. 10, No. 8, pp. 707-710).
[13] Frey, B.J. and Dueck, D., 2007. Clustering by passing messages between data points. science, 315(5814), pp.972-976.
[14] Bäck, T. and Schwefel, H.P., 1993. An overview of evolutionary algorithms for parameter optimization. Evolutionary computation, 1(1), pp.1-23.

to play the game. It had no way, however, of knowing when it should do any of these things nor how it should explore the level. This will be solved by the genetic nature of the agent. Each genetic agent will have a set of parameters, the value of which will dictate the actions of the agent. The chosen parameters are the following:

1) Objective Parameter - $\mathbf{O}_p$
2) Fighting Parameter - $\mathbf{F}_p$
3) Currency Parameter - $\mathbf{C}_p$
4) Health Item Parameter - $\mathbf{H}_p$
5) Exploration Method Parameter - $\mathbf{E}_p$

Each of these parameters can have a value between 0 and 100. The first four parameters encode the preference the genetic agent will give to reaching the objective, fighting enemies, gathering currency and gathering health items, respectively. The fifth parameter encodes the method of exploration that the agent will pursue. Different combinations of parameters will therefore lead to different behaviours from the agent even when facing the same situation.
In the next section, we will describe the evolutionary algorithm used to find the sets of parameters that will be used for the persona agents.

Evolutionary Algorithm. Having a clustering algorithm that finds the persona clusters present on a set of player traces and a genetic agent that behaves differently according to its genome, we now need a method of finding the set of parameters that dictate a behaviour that most resembles the traces found on each persona cluster. To do so, we will use an evolutionary algorithm.
For each persona cluster, we initialise a population of 300 genetic agents with random sets of parameters. At each generation, all of the agents play the same level of the “Flower Hunter” game. Their actions are recorded and compared to the action traces of the representative player trace of the persona cluster. This comparison is done using the behavioural distance metric previously described. This distance is then negated and used to assign a fitness to each of the agents: the greater the distance between the agent trace and the cluster representative trace, the lower the fitness of the agent. All fitness values will therefore be negative, with the highest possible fitness being 0, when the traces compared are exactly the same.
Once every agent has been assigned a fitness, the 30% with the highest fitness are chosen to reproduce and be a part of the next generation. The reproduction is done by randomly selecting two parent agents and randomly mixing their parameters to create a new child agent. With a 20% probability, each of the parameters can suffer a mutation and take a random value. The top 30% of the population thus reproduces between itself until the population has once again 300 agents. This process is repeated for 60 generations. The agent with the set of parameters that allow it to have the best fitness in the last population is then chosen as the persona agent.

Table 2: Persona clusters found by the clustering algorithm, along with their number of members, fitness of the corresponding persona agent, and whether the persona agent was accepted as representing the behaviour of the cluster.

Cluster Nº	*Members*	*Persona Agent Fitness*	*Accepted*
1	1	-493	No
2	10	-138	Yes
3	2	-572	No
4	1	-682	No
5	10	-348	Yes
6	1	-1214	No
7	2	-536	No
8	3	-239	Yes
9	12	-185	Yes
10	29	-75	Yes
11	1	-8846	No
12	4	-377	Yes
13	1	-589	No
14	5	-190	Yes
15	8	-192	Yes
16	1	-879	No

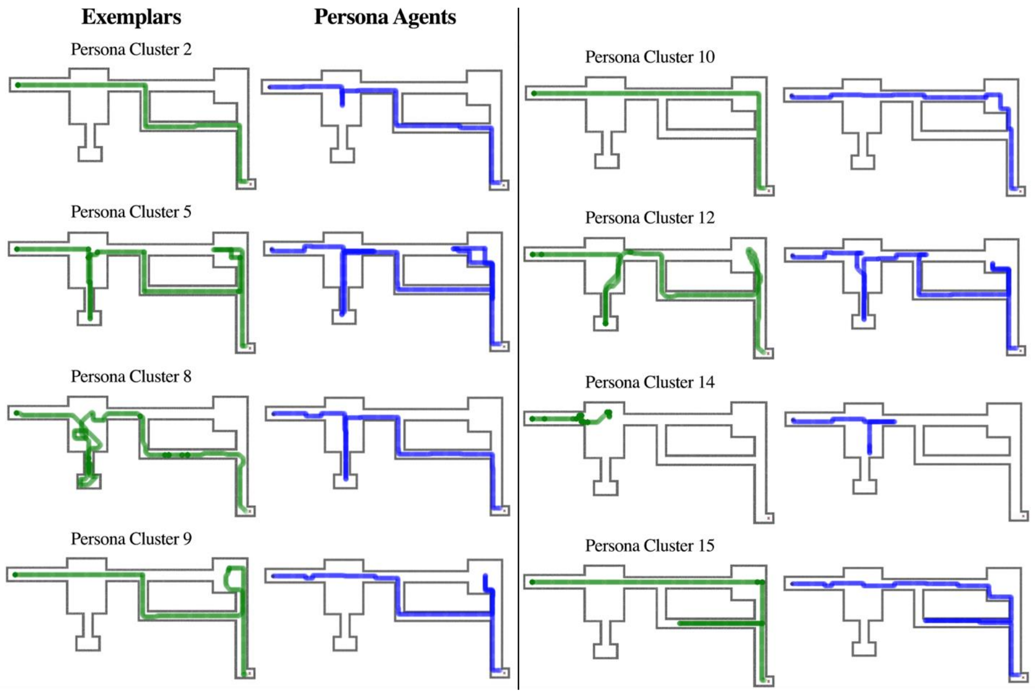


Figure 24: Representation of the game paths taken by the persona cluster representative (left, in green) and the corresponding persona agent (right, in blue).

Results. The first step of the persona agent pipeline was clustering the 91 player traces we had collected into clusters that represent different playing styles, obtaining 16 clusters. As can be seen on Table 2, some of these clusters have only one member, representing outliers that behaved in such a particular way that they could not be added to any of the other clusters. One of these players went around in circles multiple times whereas another went back and forth in the same corridor. We decided such traces should not be discarded from the data-set as they still represented player behaviour, however strange it might be. Other clusters have many members, like cluster number 10, which has 29 members and represents all players that went on a straight line to the rightmost corner of the game then straight down to the final objective (Figure 24).

Having defined the persona clusters, we then evolved the parameters of 16 persona agents so that each one of them represented the behaviour of one of the persona clusters. We thus obtained 16 persona agents, the fitness of which can be found on Table 1. Only agents with a fitness above -400 were considered acceptable as representatives of their persona cluster. This value was chosen through observation. Traces with a fitness above -400 could not be spotted as outliers when compared to the player traces of the cluster. The traces of the accepted persona agents can be seen next to the traces of the corresponding persona cluster representative on Figure 24. In the end, only half of the persona clusters were properly represented by a persona agent. However, the 8 persona agents that were accepted represented 89% of the player traces collected, as every cluster with more than 2 members was successfully represented. This was an unexpected result as the number of members in a cluster has no influence on the evolution of the agent, given the fitness is always calculated based only on the representative of the cluster.

SECTION 5 - EXAMPLE WORKFLOWS

In the previous sections, we have explained the architecture of our SETAs and the several modules that were researched and developed, both for the prediction of components of UX as well as for the generation of UX testing agent behaviour. In this section, we aim to clarify how the SETAs can be used by designers and test engineers during the development of a XR system.

SECTION 5.1 - USING FORMAL EXPERIENCE SPECIFICATIONS FOR TESTING

One application of our SETAs is the automatic evaluation of UX for a given set of already created levels or scenarios. To do this, the designer needs to connect the System Under Test (SUT) to the iv4XR framework. Once this is done, the designer can define a set of Desired Experience Outcomes (DEO) describing the experiences that are expected (or undesired) for players overall or for a specific subset of players. They can then run SETAs on the set of levels/scenarios that will be evaluated. These SETAs will be equipped with the relevant modules for the UX component being evaluated. By running the SETAs on the levels/scenarios, the resulting UX predictions can be automatically compared to the defined DEO and the designer can be informed about the results of the test. The information given to the designer can be a simple “pass” or “fail” for each level/scenario or be complemented by more information, like the heat maps shown in Section 4.1. An overview of this example workflow is shown in Figure 25.

To give a concrete example, let us imagine a designer testing an exploration game in VR. This designer has connected the VR game to the iv4XR network and now wishes to evaluate the UX of a number of levels that have been designed. Given the exploratory nature of the game, the designer decides that the most important component of UX that she wants to test is the emotional state of the player, with a special focus on the arousal level. She wants to ensure players will not become bored (low arousal) when traversing the level. The designer had already collected arousal traces in previous levels of the game using an approach similar to the one described in Section 3.1.2. This allows her to use the machine learning based emotion prediction module described in Section 3.1.2 to automatically estimate the level of arousal. She is also interested in knowing if players can feel Cybersickness when playing the game, so she also adds the module described in Section 3.3, thus having two different UX metrics modules implemented in her SETA. As she is mostly interested in knowing what users would do and already has collected traces, she decides to use the behavioural module described in Section 4.2 to find the different types of players she has and simulate their behaviour in the new levels she is testing. By defining the UX metrics and the behavioural module for the SETA, the designer has thus created the UX testing agent that best fits her system under test and testing goals.

As a formal experience specification, our designer can decide to only accept levels that have at least two peaks of arousal. This can be described as Sat(AA) according to the specification language described in Section 4.1. Having both a SETA and a formal experience specification, the designer can finally run the SETA through all the levels and have automatically tested whether the level and SETA pair passes the constraints defined in the formal experience specification or not.

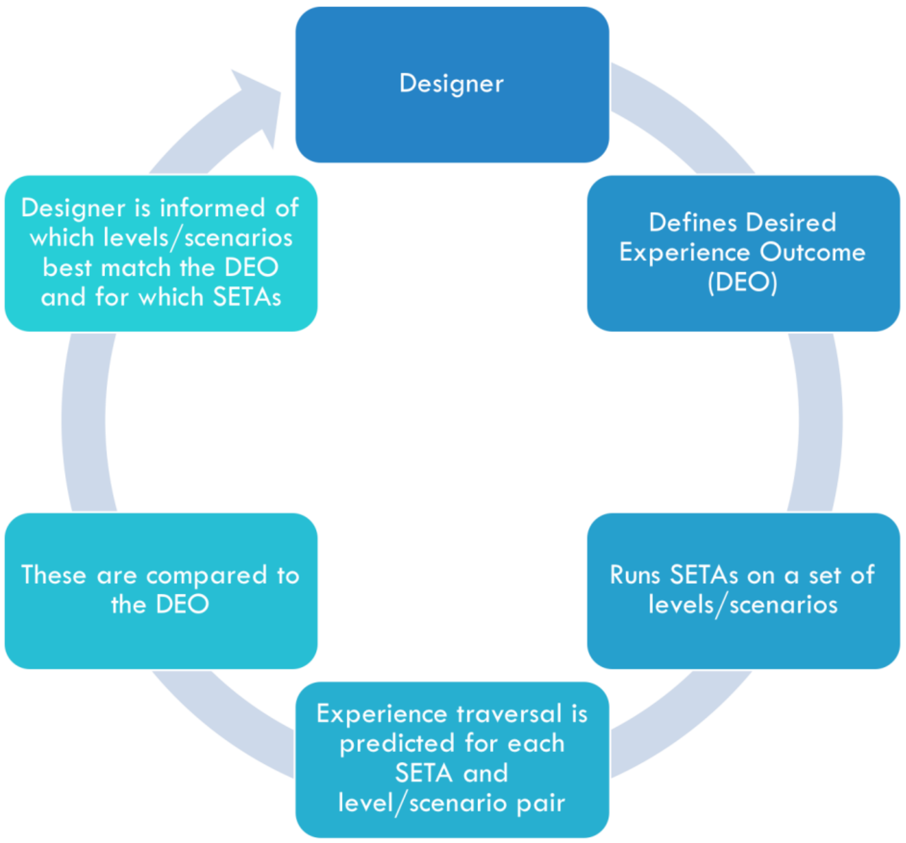

Designer
Defines Desired Experience Outcome (DEO)
Runs SETAs on a set of levels/scenarios
Experience traversal is predicted for each SETA and level/scenario pair
These are compared to the DEO
Designer is informed of which levels/scenarios best match the DEO and for which SETAs

Figure 25: Workflow for the ranking and selection of a set of levels/scenarios based on UX criteria.

SECTION 5.2 - USING UX TEST AGENTS FOR EXPERIENCE BASED PROCEDURAL CONTENT GENERATION

Another application of our SETAs can be the automatic generation of scenarios (e.g. game levels) that induce a specific desired UX. To do this, the designer needs once again to have the SUT connected to the iv4XR framework. A parameterized Procedural Content Generator (PCG) that is able to create new levels/scenarios based on a given set of parameters is also needed, and this is not directly offered by the framework as it is very SUT dependent. The iv4XR toolkit does offer, however, 2D level generators that can be modified to be used by other SUTs. The designer then needs to define the set of Desired Experience Outcomes (DEO) that describe the experiences that will guide the generation of the new levels/scenarios. A population of levels/scenarios created by the parameterized PCG engine can then be created and evolutionary algorithms used to evolve this population to best match the DEOs. The fitness of the levels/scenarios will be given by running SETAs equipped with the relevant modules for the UX component being evaluated on each level/scenario. The closer the predicted experience is to the DEOs, the better the fitness of the level/scenario and the higher the chance that this level/scenario and its offspring will remain in the population. When the evolutionary algorithm reaches its final generation, the designer will be presented with the set of levels found that best match the DEOs that had been defined. An overview of this example workflow is shown in Figure 26.

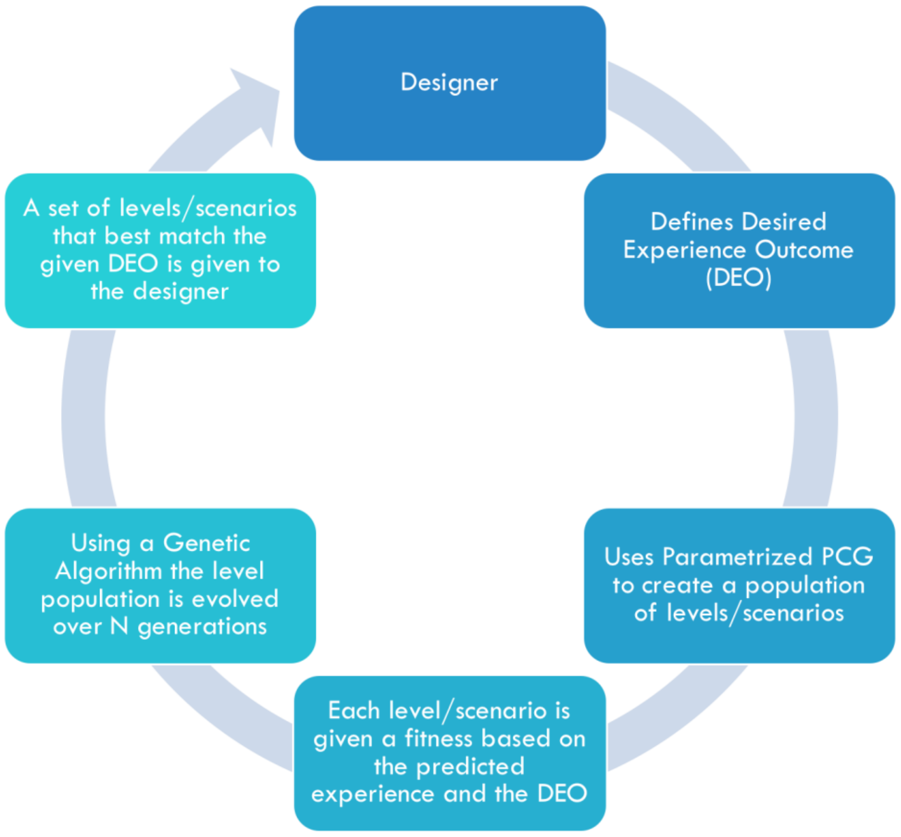

Designer
Defines Desired Experience Outcome (DEO)
Uses Parametrized PCG to create a population of levels/scenarios
Each level/scenario is given a fitness based on the predicted experience and the DEO
Using a Genetic Algorithm the level population is evolved over N generations
A set of levels/scenarios that best match the given DEO is given to the designer

Figure 27: Workflow for the automatic generation of new levels/scenarios based on a desired UX.

SECTION 6 - CONCLUSIONS

The goal of WP4 was to create Socio Emotional Test Agents (SETAs) capable of testing the UX of systems connected to the iv4XR framework. Such agents required both the ability to measure components of UX as well as the capability of behaving in ways that were aligned with the testing of UX.

In order to create such SETAs, we developed UX metrics modules capable of emotion prediction, cognitive load prediction, motion sickness prediction, difficulty estimation and validating the plot of interactive narrative games. We also developed behavioural modules that allow the SETAs to either mimic the behaviour of particular types of users or to behave in ways that try to maximise emotional coverage. We further worked on creating a specification language capable of defining UX goals so that they could be automatically tested.

Overall, we created solutions that allow designers and testers that connect their applications to the iv4XR framework to create tailored made SETAs to automatically test different components of UX.