Model-based Player Experience Testing with Emotion Pattern Verification

Saba Gholizadeh Ansari[1](✉), I. S. W. B. Prasetya[1] Davide Prandi[2], Fitsum Meshesha Kifetew[2], Mehdi Dastani[1], Frank Dignum[3], and Gabriele Keller[1]

[1] Utrecht University, Utrecht, The Netherlands, `s.gholizadehansari@uu.nl`
[2] Fondazione Bruno Kessler, Trento, Italy
[3] Umeå University, Umeå, Sweden

Abstract. Player eXperience (PX) testing has attracted attention in the game industry as video games become more complex and widespread. Understanding players' desires and their experience are key elements to guarantee the success of a game in the highly competitive market. Although a number of techniques have been introduced to measure the emotional aspect of the experience, automated testing of player experience still needs to be explored. This paper presents a framework for automated player experience testing by formulating emotion patterns' requirements and utilizing a computational model of players' emotions developed based on a psychological theory of emotions along with a model-based testing approach for test suite generation. We evaluate the strength of our framework by performing mutation test. The paper also evaluates the performance of a search-based generated test suite and LTL model checking-based test suite in revealing various variations of temporal and spatial emotion patterns. Results show the contribution of both algorithms in generating complementary test cases for revealing various emotions in different locations of a game level.

Keywords: automated player experience testing, agent-based testing, model-based testing, models of emotion

1 Introduction

Player experience (PX) testing has become an increasingly critical aspect of game development to assist game designers in realistically anticipating the experience of game players in terms of enjoyment [17], flow [46] and engagement [31]. While functional testing is intended to test the functionality of the game [38], the PX testing verifies whether emotions and psychology of players shaped during the interaction with the game are close to the design intention. This helps game designers in early development stages to identify design issues leading to game abandon, improve the general experience of players and even invoke certain experience during the game-play [53,3,25]. Let us also clarify that 'usability' is a concept in the broad domain of PX testing, but not the only concept. Usability

L. Lambers and S. Uchitel (Eds.): FASE 2023, LNCS 13991, pp. 151–172, 2023.
https://doi.org/10.1007/978-3-031-30826-0_9

tests are designed to address issues that can lead to degrading the human performance during the game-play [10], whereas PX can target the emotional experience of a player which eventually influences the success or failure of a game in the market [1]. This has led to the emergence of *Games User Research* (GUR) as an approach to gain insights into PX which is tied to human-computer-interaction, human factors, psychology and game development [14].

Validating a game design relies either on trained PX testers or acquiring information directly from players with methods such as interviews, questionnaires and physiological measurements [40,37], which are labour-intensive, costly and not necessarily representing all users profiles and their emotions. Moreover, such tests need to be repeated after every design change to assure the PX is still aligned with the design intention. Thus, GUR researchers have turned into developing AI-based PX testing methods. In particular, agent-based testing has attracted attention because it opens new rooms for automated testing of PX by imitating players while keeping the cost of labour and re-applying the tests low.

There exist appraisal theories of emotions that address the elicitation of emotions and their impact on emotional responses. They indicate that emotions are elicited by appraisal evaluation of events and situations [33]. Ortony, Clore, and Collins (**OCC**) theory [43] is one of several widely known appraisal theories in cognitive science that is also commonly used in modeling emotional agents [15,9,47,42,12]. Despite the influence of emotions on forming the experience of players [39,13], this approach has not been employed in PX testing [6].

In our automated PX testing approach, we opt for a model-driven approach to model emotions. Theoretical models of human cognition, used for decades in cognitive psychology, provide a more coherent outlook of cognitive processes. In contrast, applying a data-driven (machine learning) approach is greatly constrained by the availability of experimental data. Inferring a cognitive process from limited experimental data is an ill-posed problem [5] because such a process is subjective. Individuals can evaluate the same event differently due to age, gender, education, cultural traits, etc. For example, when a romantic relationship ends, some individuals feel sadness, others anger, and some even experience relief [48]. However, according to appraisal theories of emotions, common patterns can be found in emergence of the same emotion. These patterns are given as a structure of emotions by the aforementioned OCC. Thus, a model-driven approach derived form a well-grounded theory of emotions such as OCC, is sensible when access to a sufficient data is not possible.

In this paper, we present an agent-based player experience testing framework that allows to express emotional requirements as patterns and verify them on executed test suites generated by *model-based testing* (MBT) approach. The framework uses a computational model of emotions based on OCC theory [21] to generate the emotional experience of agent players. Comparing to [21], this paper contributes to expressing emotion patterns' requirements and generating covering test suites for verifying patterns on a game level. We show such a framework allows game designers to verify the emotion patterns' requirements and gain insight on emotions the game induce, over time and over space.

Revealing such patterns requires a test suite that can trigger enough diversity in the game behavior and as a result in the evoked emotions. This is where the model-based testing approach with its fast test suites generation can contribute. In this paper, we employ an *extended finite state machine* (EFSM) model [18] that captures all possible game play behaviors serving as a subset of human behaviors, at some level of abstraction. We use a *search based algorithm* (SB) for testing, more precisely *multi objective search algorithm* (MOSA) [44], and *linear temporal logic* (LTL) for *model checking* (MC) [8,11] as two model-based test suite generation techniques to investigate the ability of each generated test suite in revealing variations of emotion e.g absence of an emotion in a corridor. We apply test-cases distance metric to measure test suites' diversity and the distance between SB and MC test suites. Results on our 3D game case study shows that SB and MC, due to their different techniques for test generation, produce distinctive test cases which can identify different variations of emotions over space and time, that cannot be identified by just one of the test suites.

The remainder of this paper is organized as follows. Section 2 explains the computational model of emotions and the model-based testing approach. Section 3 presents the PX framework architecture. Section 4 describes our methodology of expressing PX requirements using emotion patterns, test suites diversity measurement, and the overall PX testing algorithm. Section 5 shows an exploratory case study to demonstrate the emotion pattern verification using model-based testing along with an investigation on results of SB and MC test suite generation techniques. Mutation testing is also addressed in this section to evaluate the strength of the proposed approach. Section 6 gives an overview of related work. Finally, Section 7 proposes future work and concludes the paper.

2 Preliminaries

This section summarizes the OCC computational model of emotions [21] and the model-based testing as key components of our PX framework.

2.1 Computational Model of Emotions

Gholizadeh Ansari et al. [21] introduces a transition system to model goal-oriented emotions based on a cognitive theory of emotions called OCC. The OCC theory gives a structure for 22 emotion types, viewed as cognitive processes where each emotion type is elicited under certain conditions. The structure is constructed based on the appraisal theory which is validated with a series of experiments in psychology [50,16,49]. The appraisal conditions, exist in the OCC, are modeled formally in [21] for six goal-oriented emotion types (*ety*), namely: hope, joy, satisfaction, fear, distress, and disappointment for a single agent simulations where the agent's emotional state changes only by game dynamism expressed through events to the agent. A game 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:

$$(S, s_0, G, E, \delta, Des, Thres)$$

- G is a set of goals, that the agent wants to achieve; each is a pair $\langle id, x\rangle$ of a unique id and significance degree.
- S is the set of M's possible states; each is a pair $\langle K, Emo\rangle$:
  - K is a set of propositions the agent believes to be true. It includes, for each goal g, a proposition $status(g, p)$ indicating if g has been achieved or failed, and a proposition $\mathbf{P}(g, v)$ with $v \in [0..1]$, 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 $\langle ety, w, g, t_0\rangle$, ety is the emotion type, w is the intensity of the emotion respecting a goal g, and triggered time t_0.
- $s_0 \in S$ 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 [43], but only desirability matters for aforementioned types of emotion [21].
- $Thres$: thresholds for activating every emotion type.

The **transition function** δ updates the agent's state $\langle K, Emo\rangle$, triggered by an incoming event $e \in E$ as follows:

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

- $K' = e(K) \setminus H$, where $e(K)$ is the updated beliefs of the agent when K is exposed to e; these may include updates on goals' likelihood and their status. H is the set of likelihoods of goals that are achieved or failed; this information is no longer needed and removed from $e(K)$.
- $Emo' = newEmo(K, e, G) \oplus decayed(Emo)$, where $newEmo(K, e, G)$ and $decayed(Emo)$ are *newly* activated emotions and the still active emotions that decay over time. The operator $\oplus$ merges them after applying some constraints [21].

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

$$newEmo(K, e, G) = \{\langle ety, g, w, t\rangle \mid ety \in Etype,\ g \in G, w = \mathcal{E}_{ety}(K, e, g) > 0\} \quad (1)$$

where w is the intensity of the emotion ety towards the goal $g \in G$ and t is the current system. 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 $g \in G$. $\mathcal{E}_{ety}(K, e, g)$ 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 go up which conveys they become more tolerant before feeling negative-valenced emotions. There is also a memory (*emhistory*) of activated emotions in the past for some reasonable time frame. This is maintained implicitly in the emotions' activation functions. The activation function of each emotion, based on provided definitions in the OCC theory, is as bellows, where x, v and v' refer to the goal's importance, the goal likelihood in previous and the new state respectively.

- $\mathcal{E}_{\boldsymbol{Hope}}(K, e, g) \;=\; \overbrace{\underbrace{v' * x}_{\text{activation potential}} \;-\; Thres_{\,Hope}}^{\text{activation intensity}}$
  provided $g = \langle id, x \rangle \in G$, $\mathbf{P}(g, v) \in K$, $\mathbf{P}(g, v') \in e(K)$, and $v{<}v'{<}1$.
- $\mathcal{E}_{\boldsymbol{Fear}}(K, e, g) = (1 - v') * x - Thres_{\,Fear}$, provided $g = \langle id, x \rangle \in G$, $\mathbf{P}(g, v) \in K$, $\mathbf{P}(g, v') \in e(K)$, and $0{<}v'{<}v$.
- $\mathcal{E}_{\boldsymbol{Joy}}(K, e, g) = Des(K, e, g) - Thres_{\,Joy}$, provided $g \in G$, $\mathbf{P}(g, 1) \in e(K)$, and $Des(K, e, g) > 0$.
- $\mathcal{E}_{\boldsymbol{Distress}}(K, e, g) = |Des(K, e, g)| - Thres_{\,Distress}$, provided $g \in G$, $\mathbf{P}(g, 0) \in e(K)$, and $Des(K, e, g) < 0$.
- $\mathcal{E}_{\boldsymbol{Satisfaction}}(K, e, g) = x - Thres_{\,Satisfaction}$, provided that $g = \langle id, x \rangle \in G$, $status(g, achieved) \in e(K)$, and both $\langle Hope, g \rangle, \langle Joy, g \rangle \in emhistory$.
- $\mathcal{E}_{\boldsymbol{Disappointment}}(K, e, g) = x - Thres_{\,Disappointment}$, provided $g{=}\langle id, x \rangle \in G$, $status(g, failed) \in e(K)$, and both $\langle Hope, g \rangle, \langle Distress, g \rangle \in emhistory$.

Emotion decay. An emotion intensity in Emo declines over time, triggered by *tick* events. This is formulated with a decay function over intensity as follows:

$$decayed(Emo) = \{\langle ety, g, w', t_0 \rangle \mid \langle ety, g, w, t_0 \rangle {\in} Emo,\; w' {=} \mathsf{decay}_{\mathbf{ety}}(w_0, t_0) > 0\} \quad (2)$$

where w_0 is the initial intensity of ety for the goal g at time t_0; this is stored in *emhistory*. $\mathsf{decay}_{\mathsf{ety}}$ which is a decay function defined as an inverse exponential function over the peak of intensity (w_0) at time t_0.

2.2 Model-based Testing with EFSM

Since automated testing is a major challenge for the game industry due to the complexity and hugeness of games' interaction space, a recent development is to apply a model-based approach for test generation. [30,52,18]. For this purpose, an extended finite state machine (EFSM) M can be used which is a finite state machine (FSM), extended with a set V of *context variables* that allows the machine to have richer concrete states than the abstract states of its base FSM

[2]. Transitions t in M take the form $n \xrightarrow{l/g/\alpha} n'$ where n and n' are source and destination abstract states of the transition, l is a label, g is a predicate over V that guards the transition, and α is a function that updates the variables in V.

Figure 1 shows an example of a small level in a game called *Lab recruits* [4]which is the case study of this paper as well. A *Lab recruits* level is a maze with a set of rooms and interactive objects, such as doors and buttons. A level might also contain fire hazards The player's goal is to reach the object gf0. Access to it is guarded by door_3, so reaching it involves opening the door using a button, which in turn is in a different room, guarded by another door, and so on. Ferdous et al. [18] employs a combined search-based and model-based testing for functional bug detection in this game using EFSM model (Figure 1). In the model, all interactable objects are EFSM states: doors (3), buttons (4), and the goal object gf0. For each door_i, $\mathsf{d}_i\mathsf{p}$ and $\mathsf{d}_i\mathsf{m}$ are introduced to model the two sides of the door. The model has three context variables representing the state of each door (open/close). A solid edged transition on the model is unguarded, modelling the agent's trip from one object to another without walking through a door. A dotted transition models traversing through a door when the door is open. A dashed self loop transition models pressing a button; it toggles the status of the doors connected to the pressed button. Notice that the model captures the logical behavior of the game. It abstracts away the physical shape of the level, which would otherwise make the model more complicated and prone to changes during development. Given such a model, *abstract test cases* are constructed as sequences of consecutive transitions in the model. This paper will extend the EFSM model-based testing approach [18] for player experience testing.

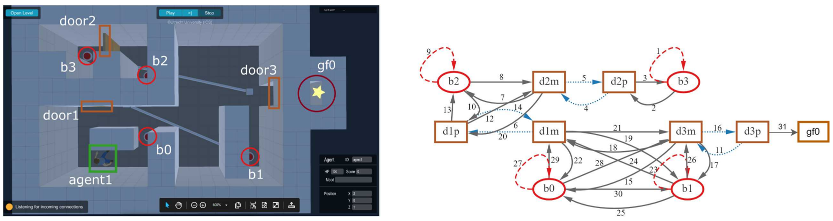


Fig. 1: A game level in the Lab Recruits game and its EFSM model [18].

3 PX Testing Framework

The proposed *automated PX testing framework* aims to aid game designers for PX assessment of their games by providing information on the time and place of emerged emotions and their patterns which would ultimately determine the general experience of player. E.g. if these patterns do not fulfill design intentions, game properties can be altered and testing process can be repeated.

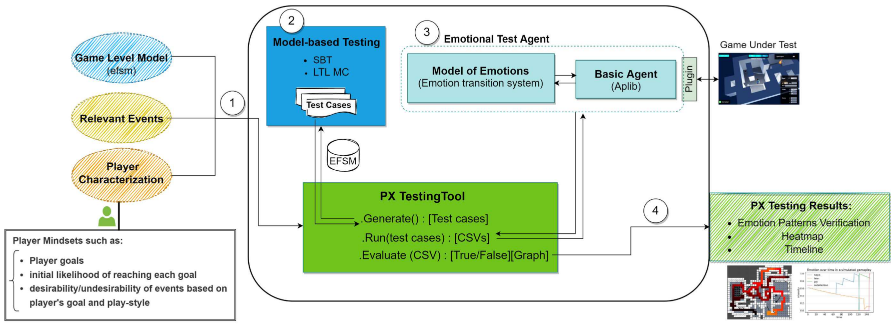
[4] https://github.com/iv4xr-project/labrecruits

Figure 2 shows the general architecture of the framework. There are four main components: a *Model-based Testing* component for generating tests, the *Model of Emotions* component implements the computational model of emotions from Section 2.1, an Aplib *basic test agent* [45] 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 these inputs, see ① in Figure 2:

- An EFSM that abstractly models the functional behavior of the game.
- A selection of game events that have impacts on the player's emotions (e.g. defeating an enemy, acquiring gold).
- 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.

Given the EFSM model, the *Model-based testing* component, ② in Figure 2, generates a test suite consisting of abstract test cases to be executed on the game under test (GUT). The test generation approach is explained in Section 4.1. Due to the abstraction of the model, emotion traces cannot be obtained from pure on-model executions. They require the executions of the test cases on the GUT. An adapter is needed to convert the abstract test cases into actual instructions for the GUT. The Aplib basic test agent does this conversion.

Attaching the *Model of Emotions* to the *basic test agent* creates an *emotional test agent*, ③ in Figure 2, 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* analyzes the traces to verify given emotional requirements and provide heat-maps and timeline graphs of emotions for the given level (④ in Figure 2).



Game Level Model (efsm)
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
1
2
Model-based Testing
SBT
LTL MC
Test Cases
EFSM
3
Emotional Test Agent
Model of Emotions (Emotion transition system)
Basic Agent (Aplib)
Plugin
Game Under Test
PX TestingTool
.Generate() : [Test cases]
.Run(test cases) : [CSVs]
.Evaluate (CSV) : [True/False][Graph]
4
PX Testing Results:
Emotion Patterns Verification
Heatmap
Timeline

Fig. 2: Automated PX testing framework architecture.

4 Methodology

This section describes the framework's model-based test generation techniques and our approach to measure a test suite's diversity. Then, our approach for expressing emotion pattern requirements and verifying them are explained.

4.1 Test Suite Generation

A test generation algorithm is applied to produce abstract test cases from a model with respect to a given coverage criterion. From now on, we refer to these abstract test cases simply as test cases. In our context, game designers can evaluate the game experience by evaluating emerging emotional experience through various paths to the game's goal. So, a proper test suite needs to cover various variations of player behavior to expose various emotion patterns. Here, we aim at graph-based coverage, such as transition coverage. However, since the model of emotions from Section 2.1 is goal-oriented, some adjustment is needed:

Definition 1. *Transition-goal coverage over an EFSM model M with respect to a goal state g is a requirement to cover all transitions in M, where a transition t is covered by a test case if its execution passes t and terminates in g.*

Given the above definition, the PX framework uses the following complementary test generation approaches; one is stochastic and the other is deterministic.

Search Based Test generation Search based testing (SBT) formulates testing problems as an optimization problem in which a search algorithm is used to find an optimized solution, in the form of a test suite, that satisfies a given test adequacy criterion encoded as a fitness function [36]. Meta-heuristic algorithms such as genetic algorithm [23] and tabu [22,26] are commonly used for this. Our framework uses an open source library EvoMBT [18] that comes with several state of the art search algorithms e.g. MOSA [44]. We utilize this to produce a test suite satisfying e.g. the criterion in Def.1 to represent players' potential behavior in the game, which are then executed to simulate their emotional experience.

To apply MOSA, individual encoding, search operators and a fitness function need to be defined. An individual I is represented as a sequence of EFSM transitions. Standard crossover and mutation are used as the search operators. MOSA treats each coverage target as an independent optimisation objective. For each transition t, the fitness function measures how much of an individual I is actually executable on the model and how close it is from covering t as in Def.1. MOSA then evolves a population that minimize the distances to *all* the targets.

LTL model checking test generation Model checking is the second technique we use for test generation. This technique is originally introduced for automated software verification that takes a finite state model of a program as an input to check whether given specifications hold in the model [8]. Such specifications can be formulated in e.g. LTL which is a powerful language for expressing system properties over time. When the target formula is violated, a model checker produces a counter example in the form of an execution trace to help debugging the model. This ability is exploited for producing test cases by encoding coverage targets as negative formulas, and converting the produced counter examples to test cases [4,11,20]. We use this to generate test suites satisfying the coverage criterion in Def.1, encoded as an LTL properties. For each transition $t : n_1 \to n_2$ in the EFSM model, the transition-goal coverage requirement to cover t is encoded as the following LTL formula:

$$\phi_t \; = \; \neg g \, \mathcal{U} \; (n_1 \; \wedge \; \mathcal{X}(n_2 \; \wedge \; \neg g \, \mathcal{U} \; g \;))$$

where g is the goal state like gf0 in Figure 1. The model checking algorithm checks whether $\neg\phi_t$ is valid on the EFSM model using depth-first traversal [29]. If it is not, a counter example is produced that visits t and terminates in g. An extra iteration is added to find the shortest covering test case.

4.2 Test Suite Diversity

Diversity is an approach to measure the degree of variety of the control and data flow in software or a game[41]. We use this approach to measure the diversity of test suites obtained from the generators in Section 4.1. A test suite's diversity degree is the average distance between every pair of distinct test cases, which can be measured in e.g. the Jaro distance metric. For a test case tc, let $\overline{tc}$ and $|\overline{tc}|$ be its string representation and its length respectively. The Jaro distance between two test cases of tc_i and tc_j is calculated as follows:

$$Dis_{\;Jaro}(\,tc_i, tc_j) \; = \begin{cases} 1 & , \text{if } m = 0 \\ 1 \; - \; \frac{1}{3}(\,\frac{m}{|\overline{tc_i}|} \; + \; \frac{m}{|\overline{tc_j}|} + \frac{m-t}{m}\,) & , \text{if } m \neq 0 \end{cases} \tag{3}$$

where m is the number of matching symbols in two strings whose distance is less than $\lfloor |\overline{tc_i}|/2 \rfloor$, assuming $\overline{tc_i}$ is the longer string; and t is half of the number of transpositions. Then, the diversity of a test suite TS is a summation of distances between every pair of distinct test cases, divided by the number such pairs:

$$Div_{avg}(TS) \; = \; \frac{\sum_{i=1}^{|TS|} \; \sum_{j=i+1}^{|TS|} Dis_{\;Jaro}(tc_i, tc_j)}{\frac{|TS| \; * \; (|TS|-1)}{2}} \tag{4}$$

where $|TS|$ is TS' size. Additionally, if TS_1 and TS_2 are two test suites, the average distance between them is:

$$Dis_{\;avg}(TS_1, TS_2) = \frac{\sum_{tc_i \in TS_1, tc_j \in TS_2} Dis_{\;Jaro}(tc_i, tc_j)}{|TS_1| \; * |TS_2|} \tag{5}$$

This is used in Section 5 to measure the distance between the test suites generated by the two approaches (Section 4.1) provided by our framework, along with their complementary effects on revealing different emotion patterns.

4.3 Emotion Patterns' Requirements and Heat-maps

In Section 2.1, we described the emotion model of an agent. When the agent executes a test case, it produces a trace of its emotion state over time. Such a *trace* is a sequence of tuples (t, p, Emo) where t is a timestamp, Emo is the agent emotion state at time t, and p is its position. Running a test suite produces a set of such traces.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*, or lack of a certain emotion type.

Definition 2. *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.*

A single pattern such as F represents the stimulation of the corresponding emotion, in this case fear. We will restrict ourselves to simply mean that this stimulation occurs, without specifying e.g. when it happens exactly, nor for how long it is sustained. A negative single pattern such as $\neg F$ represents the absence of stimulation, in this case fear. A pattern is a sequence of one or more single patterns, specifying in what order the phenomenon that each single pattern describes is expected to occurs. Patterns provide a simple, intuitive, but reasonably expressive way to express PX. 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 $J\neg FS$ when there is no stimulation of fear between J and S. The presence of this pattern indicates the presence of a 'sneak' route, where a goal is achievable without the player has to fight enough for it.

As a part of PX *requirements*, developers might insist on presence or absence of certain patterns. More precisely, given a pattern p, we can pose these types of requirements: $Sat(p)$ requires that at least one execution of the game under test satisfies p; $UnSat(p)$ requires that $Sat(p)$ does not hold; and $Valid(p)$ requires that all executions satisfy p. In the context of testing, we will judge this by executions of the test cases in the given test suite TS.

Heat-maps Whereas above we discuss emotion patterns over *time*, a heat-map shows patterns over *space*. Assuming the visitable parts of a game level form a 2D surface, we can divide it into small squares of $u{\times}u$. Given a position p and a square s, we can check if $p{\in}s$. Given a trace τ, let $Emo(s) = \{Emo \mid (t, p, Emo) \in \tau,\ p{\in}s\}$: the set of emotions, that occur in the square s. This can be aggregated by a function $aggr$ that maps $Emo(s)$ to $\mathbf{R}$. An example of an aggregator is the function max_e that calculates the maximum of a specific emotion e (e.g. hope). Section 5 will show some examples. Such maps can be analyzed against requirements, e.g. that the aggregate values in certain areas should be of a certain intensity. We can also create 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. Finally, the overall methodology of our PX testing is summarized in Algorithm 1.

4.4 PX Framework Implementation

The test agent is implemented using APlib Java library [45]. It has a BDI architecture [27] with a novel goal and tactical programming layer. We use JOCC library [21] for modeling emotions. To facilitate the model-based testing, we integrate EvoMBT[18]. It generates abstract test suites from an EFSM model, utilizing EvoSuite [19] for search-based test generation. An implementation of LTL model checking algorithm is employed to produce model checking-based test suites. The framework and its data will be available for public use.

Algorithm 1 *The Execution of automated PX Testing Algorithm.*

Input: EFSM M, coverage criterion C, configuration parameters $Config$ for test generator, and emotion pattern requirements' list R.
Output: Emotion traces, Heat-maps of emotions, and the verification of requirements' ($true/false$).

```
1: procedure EXEC($M, C, Config, R$)
2:     $TS_{abstract} \leftarrow$ TSGENERATE($M, C, Config$)
3:     $TS_{concrete} \leftarrow$ TRANSLATE($TS_{abstract}$)
4:     Configure an emotional test agent $A$
5:     $traces_{emotion} \leftarrow \emptyset$
6:     for all test cases $tc \in TS_{concrete}$ do
7:         $\tau \leftarrow A$ executes $tc$ on the SUT
8:         $traces_{emotion} \leftarrow traces_{emotion} \cup \{\tau\}$
9:     end for
10:    $Hmaps \leftarrow$ GENERATEHEAT-MAPS($traces_{emotion}$)
11:    $Vresult \leftarrow \{ (r, \text{VERIFY}(r)) \mid r \in R \}$
12:    return ($traces_{emotions}, Hmaps, Vresults$)
13: end procedure
```

5 Case Study

This section presents an exploratory case study conducted to investigate the use of a model-based PX testing framework[5] for verifying emotion requirements in a game-level and to investigate the difference between the search based generated test suite and the model checker generated test suites on revealing emotion patterns. Finally, we run mutation testing to evaluate the strength of our framework.

5.1 Experiment Configuration

Figure 3 shows a test level called *Wave-the-flag* in the Lab Recruits, a configurable 3D game, designed for AI researchers to define their own testing problems. It is a medium sized level, consisting of a 1182 m^2 navigable virtual floor, 8 rooms, 12 buttons, and 11 doors. Its EFSM model consists of 35 states and 159 transitions. The player starts in the room marked green at the top, and must find a 'goal flag' gf0 marked red in the bottom room to finish the level. Doors and buttons form a puzzle in the game. A human player needs to disclose the connections between buttons and doors to open a path through

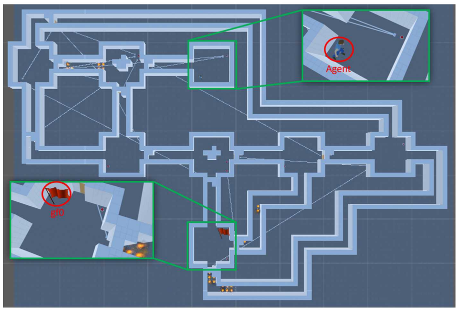


Fig. 3: *Wave-the-flag* level.

[5] https://doi.org/10.5281/zenodo.7506758

the maze to reach the aforementioned goal flag in a timely manner. The player can earn points by opening doors and lose health in case of passing fire flames. For the test agent, the latter is also observable as an event called *Ouch*. If the player runs out of health, it loses the game. The player also has no prior knowledge about the position of doors, buttons and the goal flag, nor the knowledge on which buttons open which doors. Since there are multiple paths to reach the target, depending on the path that the player chooses to explore, it might be able to reach the goal without health loss, at one end of spectrum, or it can end up dead at the other end. The EFSM model (not shown) of the *Wave-the-flag* level is constructed similar to the running example in section 2.2. To add excitement, *Wave-the-flag* also contains fire flames. However, these flames are not included into the EFSM model because the placement and amount of these objects are expected to change frequently during development. Keeping this information in the EFSM model would force the designer to constantly update the model after each change in flames. Thus, similar to the running example, the EFSM model contains doors, buttons and goal flags.

In addition to the EFSM model, we need to characterize a player to do PX testing (① in Figure 2). Table 1 shows basic characteristics of a player, defined with a set of parameters, to configure the emotion model of the agent before the execution. The level designer determines values of these parameters. After the execution of the model, we asked the designer to check the plausibility their values by checking the emotional heat-map results. The designer checked randomly selected number of test cases with their generated emotional heat maps to check the occurrence of emotions are reasonable. Thus, the utilized values for the following experiment is confirmed reasonable by the designer. Moreover, The likelihood of reaching the goal `gf0` is set to 0.5 in the initial state to model a player who initially feels unbiased towards the prospect of finishing the level. Thus, the agent feels an equal level w of hope and fear at the beginning.

5.2 PX Testing Evaluation

Test suites are generated from the EFSM model using LTL model checking (MC) and the search-based (SB) approach with the full transition-goal coverage criterion (Def.1) named as TS_{SB} and TS_{MC}, both with 60 seconds time budget.

Abstract test suite characteristics. Our reason for using multiple test generation algorithms is to improve the diversity of the generated test cases, which in turn would improve our ability to reveal more emotion patterns. Table 2 shows the basic characteristics of the generated test suites. Due its stochastic behavior, the search-based (SB) generation is repeated 10 times, and then averaged. The SB algorithm manages to provide full transition-goal coverage with, in average, 54.6 test cases ($\sigma = 7.8$), with the average diversity of 0.192 ($\sigma = 0.03$) between test cases in a test suite. The model checker (MC) always satisfies the criterion with 74 test cases and average diversity of 0.113. The higher diversity of SB test suites (TS_{SB}) can be explained through the stochastic nature of the search algorithm. Table 2 also shows the length of the shortest and longest test cases. While SB manages to find a shorter test case with only 17.7 transitions

Table 1: *Configuration of Player Characterization. G is the agent's goal set; it has one goal for this level, namely reaching the goal-flag $gf0$, s_0 is the emotion model's initial state, a set of relevant events (E) needs to be defined by the designers: DoorOpen event, triggered when a new door gets open, is perceived as increasing the likelihood of reaching $gf0$ by v_1 in the model, Ouch event, that notifies fire burn, is perceived as declining the likelihood of reaching $gf0$ by v_2, GoalInSight event, triggered at the first time the agent observes the goal $gf0$ in its vicinity , is modelled as making the agent believes that the likelihood of reaching the goal becomes certain* (1)*, and finally GoalAccomplished event is triggered when the goal $gf0$ is accomplished. Des reflects the desirability/undesirability of each event with respect to the goal and Thres is the emotions' activation thresholds. x, v_i, and y_i are constants determined by the designer.*

Parameter	**Value**
G	$g = < gf0, x > \in G$
s_0	$likelihood(gf0, 0.5) \in K_0$, $Emo_0 = \{< Hope, gf0, w, 0 >, < Fear, gf0, w, 0 >\}$
E	$= \{DoorOpen, Ouch, GoalInSight, GoalAccomplished\}$ on $DoorOpen$ event: $likelihood(gf0, +v_1)$, on fire burn in $Ouch$ event: $likelihood(gf0, -v_2)$, on $GoalInSight$ event: $likelihood(gf0, 1)$.
Des	$Des(K, DoorOpen, gf0) = +y_1$, $Des(K, Ouch, gf0) = -y_2$, $Des(K, GoalInSight, gf0) = +y_3$
$Thres$	0

in average, its longest test case has in average 74.25 transitions. Finally, the last row in Table 2 indicates the difference between SB and MC test suites. The distance between two test suites is measured for every generated TS_{SB} using Equation 5 which brings about 0.214 ($\sigma = 0.024$) distance in average between test cases of the two suites. Later, we investigate whether such a difference can lead to differences at the execution level in emotion patterns.

Table 2: *Characteristics of LTL-model checking-based and search-based test suites with respect to the same coverage criterion.*

Test suite	**size**	$Div_{avg}(TS_i)$	**Shortest** tc	**longest** tc
TS_{MC}	74	0.113	23	45
$TS_{SB_{avg}}$	54.6	0.192	17.7	74.25
$Div\ _{avg}(TS_{MC}, TS_{SB})$	128.6	0.214		

Evaluation of emotional heat-maps. Inspecting the emerging emotions requires real execution of test cases on the game under test. The execution of TS_{MC} with 74 test cases and the TS_{SB} with the average 54.5 test cases took 11,894 seconds and 10,201 respectively in the game. After the executions, the automated PX testing framework produces a heat-map of emotions for every test case to give spatial information about the intensity of the emotion at each location in the game. Unlike [21] which only produces heat-maps of emotions for a single pre-defined navigation path, Figure 4 shows the *aggregated* heat map visualization of some selected emotions, evoked during the execution of *all* test cases in TS_{MC} and a randomly chosen TS_{SB} suite from the previously generated 10 TS_{SB} suites, with the square size u=1 and max_e as the aggregation function. So, the maps show the maximum intensity on a given spot over the whole execution of the corresponding test suite. The brighter color shows the higher intensity of an emotion. In this case, the bright yellow represents the highest emotional intensity in heat maps. The heat maps of hope, joy and satisfaction for these test suites show quite similar spatial information (only hope and joy are shown in Figure 4). However, TS_{MC} generally shows a higher level of hope during the game-play (Figures 4a and 4b). So, if the designer verifies his level on the presence and spatial distribution of intensified hope through the level, the test cases produced by TS_{MC} can expose these attributes better. This can be explained by the model checker setup to find shortest test cases; some can then open the next door sooner, raising hope before its intensity decays too much.

The maps also show a difference in the spatial coverage of TS_{SB} and TS_{MC} (marked green in Figures 4a and 4b). The transition that traverses the corridor is present in TS_{MC}, but when the corresponding abstract test case is transformed into an executable test case for APlib test agent, they also incorporate optimization. So, it finds a more optimized way for execution by skipping the transition that actually passes the corridor towards the room, if the next transition is to traverse back along same corridor. The corridor is, however, covered by TS_{SB}.

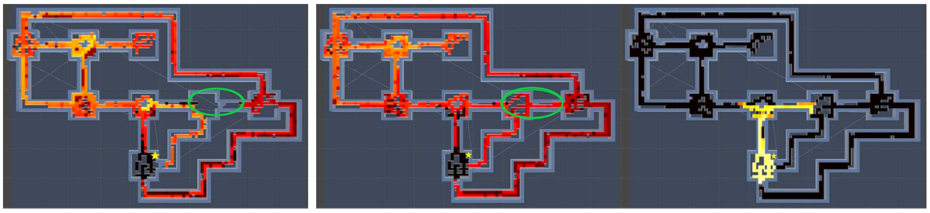
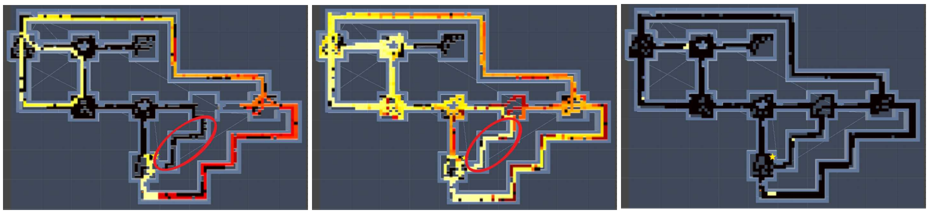

(a) MC-Hope (b) SBT-Hope (c) SBT-Joy

Fig. 4: Heat-map visualization of positive emotions for SBT and MC test suites.

The most striking differences between TS_{SB} and TS_{MC} are revealed in their negative emotions' heat-maps (Figure 5). Most places that are marked black as distress-free by executed TS_{MC} (Figure 5a) are actually highly distressful positions for some test cases of TS_{SB}. The presence of distress might be the intended player experience, whereas its absence in certain places might actually be undesirable. Upon closer inspection of individual test cases, it turns out that the test cases of TS_{SB} that pass e.g. the red regions in Figure 5a and 5b always show distress in the marked corridor, whereas one test case in TS_{MC} manages to find a 'sneak route' that passes the corridor without distress, and finishes the level successfully. Thus, if the designer is looking for the possibility of absence of distress in the sneak corridor, inspection of TS_{SB} would not suffice. The heat-maps of disappointment reveals another difference. While TS_{MC} only finds one location where the agent dies and feels disappointed, TS_{SB} manages to find 3 more locations in the level with the disappointment state (Figure 5c).

The main reason behind those differences is that a sequence of transitions results in experiencing an emotion in the agent, not just a single transition. Furthermore, emotions intensity has a residual behavior which means a sequence of transitions and behavior might result in an emotion which still remains in the agent emotional state after some time. Thus, providing state coverage or the transition-coverage criterion does not in itself suffice to manage revealing possible emotions and their patterns. The variation of transitions and their order in a test case can resemble the different player behaviors during the game-play that their outcomes ultimately form the player emotional experience. Therefore, finding a proper test suite that can capture the distributions of theses emotions with test cases exhibiting the presence or absence of emotions in various locations is challenging. As remarked before, due to the stochastic nature of its algorithm, the search algorithm produces more diverse test suite than the LTL model checker, and hence can increase the chance of revealing more variation of emotions in different locations of the level. However, our experiments show the model checker does provide useful complementary test cases, e.g. for finding corner cases which can be covered only by the model checker that were missed by SB. All mentioned differences can explain the 0.20 distant difference between TS_{MC} and TS_{SB}.



(a) MC-Distress (b) SBT-Distress (c) SBT-Disappointment

Fig. 5: Heat-map visualization of negative emotions for SBT and MC test suites.

Checking emotion pattern requirements. The PX testing framework is also capable of verifying emotion requirements using patterns as defined in Definition 2 format based on stimulation of emotions. These patterns are verified by inspecting the order in which different emotions are stimulated, as recorded in the trace files. Although there are numerous combinations of emotions, only some of them matter 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 goal. 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 insist on $UnSat(DHS)$ instead.We have collected a number of emotion pattern requirements from the designer of the *Wave-the-flag* level; these are shown in the upper part of Table 3. 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 to get bored by interpreting boredom as absence of active emotions in the agent emotional state for some time. As can be seen in Table 3, while most requirements are verified during the test, there are requirements like $Sat(J\neg S)$ that are failed. This requirement indicates the designer expects at least one execution path that joy is stimulated at least once thought the execution, but the agent never reaches the goal with satisfaction. Having *Sat* patterns failed to be witnessed, or *UnSat* patterns that are witnessed, assist the designer to alter their game level and run the agent through the level again. For example, here, the fail on $Sat(J\neg S)$ is an indication that the designer needs to put some challenging objects like fire or enemies in the vicinity of the goal $gf0$.

Table 3: *Emotion pattern check with* TS_{MC} *and* TS_{SB}*.* H*= hope,* F*= fear,* J*= joy,* D*= distress,* S*= satisfaction,* P *= disappointment and* $\neg X$ *= absence of emotion* X*.*

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

Emotion patterns length=2	101/144 (70.2%)	101/144 (70.2%)
Emotion patterns length=3	88/150 (58.6%)	88/150 (58.6%)
Emotion patterns length=4	71/164 (43.2%)	72/164 (43.9%)
Emotion pattern length=5	60/177 (33.8%)	61/177 (34.4%)

Table 3 also shows the similar ratio of the pairwise combination of emotions over various $Sat(p)$ for the pattern p between length 2-5 by the TS_{SB} and the TS_{MC}, indicating that both test suites can perform well to detect *Sat*-type emotion patterns. However, there the last three patterns in Table 3 are covered by TS_{SB} but missed by TS_{MC}. Thus, they are complementary, which makes it reasonable to use both test suites for verifying emotion pattern requirements.

5.3 Mutation Testing Evaluation

Mutation testing [32] is a technique to evaluate the quality of test suites in detecting faults, represented by faulty variants ('mutants') of the target program generated through a set of mutation operators. Here, we use this to evaluate the strength of our PX testing approach. In the procedure, we use a corrected *Wave-the-flag* level ('original' level), satisfying all the emotion pattern requirements we posed in Table 3. Mutations are applied on the original's level definition file to produce mutants (one mutation per mutant). An example of a mutation is to remove all fire flames from a certain zone in the level; Table 4 lists the used mutation operators. A mutant represents an alternate design of the level, maintaining the level's logic, but may induce different PX. To apply the mutations, the game level is divided to 16 zones of about equal size. We apply the mutation operators on each zone. Every mutant is labeled with the applied mutation operator and z_x_y where (x, y) specifies the bottom-left corner of the zone on which the mutation is applied. After dropping mutations that do not change the level's properties, we obtain 20 distinct mutants, from which we randomly choose 10 mutants for execution. We re-run both TS_{MC} and TS_{SB} test suites on each mutant. A mutant is automatically killed when the correctness of a specification is judged differently from the original results. Table 5 shows that 8 of the 10 randomly selected mutants are killed. Remaining mutants are not killed because emotion requirements might not be distinctive enough to kill them too.

Table 4: *Mutation operators*

Code	Description
RF	Remove fire
RW2WF	Relocate fire between walls
RMRF	Relocate fire in middle of a room
AMRF	Add fire in middle of a room
AW2WF	Add fire between walls

Table 5: *Kill matrix of the mutants.*

Emotion patterns	Original	RF_z_0_51	RF_z_48_17	RF_z_0_51	RMRF_z_24_0	RW2WF_z_0_34	RW2WF_z_24_51	RW2WF_z_48_0	RW2WF_z_48_17	AMRF_z_24_51	AW2WF_z_72_34
$Sat(\neg DS)$	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
$UnSat(\neg FS)$	✔	✗	✔	✔	✔	✔	✔	✔	✔	✔	✔
$Sat(J\neg S)$	✔	✔	✔	✗	✔	✔	✗	✗	✔	✗	✗
$UnSat(JD)$	✔	✔	✔	✔	✔	✔	✔	✔	✗	✔	✔
$Sat(JFS)$	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
$Sat(DHP)$	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
$Sat(DHS)$	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
$Sat(DH\neg DS)$	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
$Sat(FDHFJ)$	✔	✗	✔	✔	✔	✔	✔	✔	✔	✔	✔
$Sat(HFDDDHFJ)$	✔	✗	✔	✔	✔	✔	✔	✗	✔	✔	✔
$Sat(FDDHFP)$	✔	✗	✗	✗	✔	✔	✔	✔	✗	✔	✔

Threat to Validity. The designed character in Player Characterization, the selected coverage criterion for test generation to verify $UnSat$ specifications, and the small number of mutation testing assessments due to the computational cost are internal threats to the validity of the work. In terms of external threats, performing the experiment on one level is not safe to be generalized.

6 Related Work

A number of research has been conducted on automated play testing to reduce the cost of repetitive and labor-intensive functional testing tasks in video games [35,54]. In particular, agent based testing has been a subject of recent research to play and explore the game space on behalf of human players for testing purposes. Ariyurek et al. [7] introduces Reinforcement Learning (RL) and Monte Carlo Tree Search (MCTS) agents to detect bugs in video games automatically. Stahlke et al. [51] presents a basis for a framework to model player's memory and goal-oriented decision-making to simulate human navigational behavior for identifying level design issues. The framework creates an AI-agent that uses a path finding heuristic to navigate a level, optimized by a given player characteristics such as level of experience and play-style. Zhao et al. [55] intend to create agents with human-like behavior for balancing games based on skill and play-styles. These parameters are measured using introduced metrics to help training the agents in four different case studies to test the game balance and to imitate players with different play-styles. Gordillo et al. [24] addresses the game state coverage problem in play-testing by introducing a curiosity driven reinforcement learning agent for a 3D game. The test agent utilizes proximal policy optimization (PPO) with a curiosity factor reflected on the RL reward function with frequency of a game state visit. Pushing the agent to have the exploratory behaviour provides a better chance to explore unseen states for bugs.

Among game model-based testing, Iftikhar et al. [30] applies it on *Mario Brothers* game for functional testing. The study uses UML states machine as a game model for test case generation which manages to reveal faults. Ferdous et al. [18] employs combined search-based and model-based testing for automated play-testing using an EFSM. Search algorithms are compared regarding the model coverage and bug detection. Note that while an EFSM provides paths through a game, it can not reveal the experience of a player who navigates the path.

Despite some research on modeling human players and their behavior in agents for automated functional play testing, there are a few research on automation of PX evaluation. Holmgard et al. [28] propose to create procedural personas or player characteristics for test agent to help game designers to develop game contents and desirable level design for different players. The research proposes to create personas in test agents using MCTS with evolutionary computation for node selection. The result on *MiniDungeons 2* game shows how different personas brings about different behavior in response to game contents which can be seen as different play-styles. Lee et al. [34] investigate a data-driven cognitive model of human performance in moving-target acquisition to estimate the game difficulty for different players with different skill level. There is limited research on the emotion prediction and its usage for automation of PX evaluation. Gholizadeh et al. [21] introduce an emotional agent using a formal model of OCC emotions and propose the potential use of such an agent for PX assessment. However, the approach lacks automated path planning and reasoning, and hence it cannot do automated gameplay. Automatic coverage of game states and collecting all emerging emotions are thus not supported which are addressed in this paper.

7 Conclusion & Future work

This paper presented a framework for automated player experience testing, in particular automated verification of emotion requirement, using a computational model of emotions and model-based test generation targeting a subset of human players' behaviors. We presented a language for emotion patterns to capture emotion requirements. We also investigated the complementary impact of different test generation techniques on verifying spatial and temporal emotion patterns.

Future work. The explained language is able to capture complex patterns with the temporal order of emotions' stimulations in the framework. However, it cannot capture spatial behavior of emotions, such as differences in the heatmaps. Generally, combining spatial and temporal aspects to verify emotion requirements in specific areas and time intervals would give a more refined way to assess the emotional experience. How to capture this into formal patterns is still an open question. Investigating the application of our approach in empirical case studies with human players is future work.

References

1. Agarwal, A., Meyer, A.: Beyond usability: evaluating emotional response as an integral part of the user experience. In: CHI'09 Extended Abstracts on Human Factors in Computing Systems, pp. 2919–2930. ACM New York, NY, USA (2009)
2. Alagar, V., Periyasamy, K.: Extended finite state machine. In: Specification of software systems, pp. 105–128. Springer (2011)
3. 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)
4. Ammann, P.E., Black, P.E., Majurski, W.: Using model checking to generate tests from specifications. In: Proceedings second international conference on formal engineering methods (Cat. No. 98EX241). pp. 46–54. IEEE (1998)
5. Anderson, J.R.: Arguments concerning representations for mental imagery. Psychological review **85**(4), 249 (1978)
6. Ansari, S.G.: 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 (2020)
7. Ariyurek, S., Betin-Can, A., Surer, E.: Automated video game testing using synthetic and humanlike agents. IEEE Transactions on Games **13**(1), 50–67 (2019)
8. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
9. Bartneck, C.: Integrating the occ model of emotions in embodied characters. In: Proceeding of the Workshop on Virtual Conversational Characters: Applications, Methods, and Research Challenges (2002)
10. Bevan, N.: What is the difference between the purpose of usability and user experience evaluation methods. In: Proceedings of the Workshop UXEM. vol. 9, pp. 1–4. Citeseer (2009)
11. Callahan, J., Schneider, F., Easterbrook, S., et al.: Automated software testing using model-checking. In: Proceedings 1996 SPIN workshop. vol. 353. Citeseer (1996)
12. Demeure, V., Niewiadomski, R., Pelachaud, C.: How is believability of a virtual agent related to warmth, competence, personification, and embodiment? Presence **20**(5), 431–448 (2011)
13. Desmet, P., Hekkert, P.: Framework of product experience. International journal of design **1**(1) (2007)
14. Drachen, A., Mirza-Babaei, P., Nacke, L.E.: Games user research. Oxford University Press (2018)
15. Elliott, C.D.: The affective reasoner: a process model of emotions in a multiagent system. Ph.D. thesis, Northwestern University (1992)
16. Ellsworth, P.C., Smith, C.A.: From appraisal to emotion: Differences among unpleasant feelings. Motivation and emotion **12**(3), 271–302 (1988)
17. 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)
18. Ferdous, R., Kifetew, F., Prandi, D., Prasetya, I., Shirzadehhajimahmood, S., Susi, A.: Search-based automated play testing of computer games: A model-based approach. In: International Symposium on Search Based Software Engineering. pp. 56–71. Springer (2021)
19. Fraser, G., Arcuri, A.: Evosuite: automatic test suite generation for object-oriented software. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. pp. 416–419 (2011)

20. Gargantini, A., Heitmeyer, C.: Using model checking to generate tests from requirements specifications. In: Software Engineering—ESEC/FSE'99. pp. 146–162. Springer (1999)
21. Gholizadeh Ansari, S., Prasetya, I.S.W.B., Dastani, M., Dignum, F., Keller, G.: An appraisal transition system for event-driven emotions in agent-based player experience testing. In: Engineering Multi-Agent Systems: 9th International Workshop, EMAS 2021, Virtual Event, May 3–4, 2021, Revised Selected Papers. pp. 156–174. Springer Nature (2021)
22. Glover, F.: Tabu search—part i. ORSA Journal on computing **1**(3), 190–206 (1989)
23. Goldberg, D.E.: Genetic algorithms. Pearson Education India (2006)
24. Gordillo, C., Bergdahl, J., Tollmar, K., Gisslén, L.: Improving playtesting coverage via curiosity driven reinforcement learning agents. arXiv preprint arXiv:2103.13798 (2021)
25. 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)
26. Harman, M., Jones, B.F.: Search-based software engineering. Information and software Technology **43**(14), 833–839 (2001)
27. Herzig, A., Lorini, E., Perrussel, L., Xiao, Z.: BDI logics for BDI architectures: old problems, new perspectives. KI-Künstliche Intelligenz **31**(1) (2017)
28. Holmgård, C., Green, M.C., Liapis, A., Togelius, J.: Automated playtesting with procedural personas through mcts with evolved heuristics. IEEE Transactions on Games **11**(4), 352–362 (2018)
29. Holzmann, G.J.: The model checker spin. IEEE Transactions on software engineering **23**(5), 279–295 (1997)
30. Iftikhar, S., Iqbal, M.Z., Khan, M.U., Mahmood, W.: An automated model based testing approach for platform games. In: 2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS). pp. 426–435. IEEE (2015)
31. 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)
32. Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. IEEE transactions on software engineering **37**(5), 649–678 (2010)
33. Lazarus, R.S., Folkman, S.: Stress, appraisal, and coping. Springer publishing company (1984)
34. Lee, I., Kim, H., Lee, B.: Automated playtesting with a cognitive model of sensorimotor coordination. In: Proceedings of the 29th ACM International Conference on Multimedia. pp. 4920–4929 (2021)
35. Lewis, C., Whitehead, J., Wardrip-Fruin, N.: What went wrong: a taxonomy of video game bugs. In: Proceedings of the fifth international conference on the foundations of digital games. pp. 108–115 (2010)
36. McMinn, P.: Search-based software test data generation: a survey. Software testing, Verification and reliability **14**(2), 105–156 (2004)
37. Mirza-Babaei, P., Nacke, L.E., Gregory, J., Collins, N., Fitzpatrick, G.: How does it play better? exploring user testing and biometric storyboards in games user research. In: Proceedings of the SIGCHI conference on human factors in computing systems. pp. 1499–1508 (2013)
38. Myers, G.J., Sandler, C., Badgett, T.: The art of software testing. John Wiley & Sons (2011)

39. Nacke, L., Lindley, C.A.: Flow and immersion in first-person shooters: measuring the player's gameplay experience. In: Proceedings of the 2008 conference on future play: Research, play, share. pp. 81–88 (2008)
40. Nacke, L.E.: Games user research and physiological game evaluation. In: Game user experience evaluation, pp. 63–86. Springer (2015)
41. Nikolik, B.: Test diversity. Information and Software Technology **48**(11), 1083–1094 (2006)
42. Ochs, M., Pelachaud, C., Sadek, D.: An empathic virtual dialog agent to improve human-machine interaction. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1. pp. 89–96 (2008)
43. Ortony, A., Clore, G., Collins, A.: The cognitive structure of emotions. cam (bridge university press. Cambridge, England (1988)
44. Panichella, A., Kifetew, F.M., Tonella, P.: Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. IEEE Transactions on Software Engineering **44**(2), 122–158 (2017)
45. 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)
46. 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)
47. Reilly, W.S.: Believable social and emotional agents. Tech. rep., Carnegie-Mellon Univ Pittsburgh pa Dept of Computer Science (1996)
48. Roseman, I.J., Smith, C.A.: Appraisal theory. Appraisal processes in emotion: Theory, methods, research pp. 3–19 (2001)
49. Roseman, I.J., Spindel, M.S., Jose, P.E.: Appraisals of emotion-eliciting events: Testing a theory of discrete emotions. Journal of personality and social psychology **59**(5), 899 (1990)
50. Smith, C.A., Ellsworth, P.C.: Patterns of cognitive appraisal in emotion. Journal of personality and social psychology **48**(4), 813 (1985)
51. 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)
52. Utting, M., Legeard, B.: Practical model-based testing: a tools approach. Elsevier (2010)
53. 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)
54. Zarembo, I.: Analysis of artificial intelligence applications for automated testing of video games. In: ENVIRONMENT. TECHNOLOGIES. RESOURCES. Proceedings of the International Scientific and Practical Conference. vol. 2, pp. 170–174 (2019)
55. 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)