Expressive Response Curves: Testing Expressive Game Feel with A*

Nic Junius, Elin Carstensdottir

University of California, Santa Cruz
njunius@ucsc.edu, ecarsten@ucsc.edu

Abstract

Designing AI models for expressive character behavior is a considerable challenge. Such models represent a massive possibility space of individual behaviors and sequences of different character expressions. Iterating on designs of such models is complex because the possibility spaces they afford are challenging to understand in their entirety and map intuitively onto a meaningful experience for a user. Automated playtesting has primarily been focused on the physical spaces of game levels and the ability of AI players to enact personas and complete tasks within those levels. However, core principles of automated playtesting can be applied to expressive models to expose information about their expressive possibility space. We propose a new approach to automated playtesting for AI character behaviors: Expressive Response Curves (ERC). ERC allows us to map specific actions taken by a player to perform a particular expression to understand the affordances of an expressive possibility space. We present a case study applying ERC to Puppitor rulesets. We show that using this method we can compile paths through Puppitor rulesets to map them and further understand the nature of the expressive spaces afforded by the system. We argue that by using ERC, it is possible to give designers more nuanced information and guidance to create better and more expressive AI characters.

Introduction

Designing expressive character behavior for AI performance is a difficult undertaking thanks to the complexity of the possibility spaces they create. The scale and complexity of these expressive possibility spaces makes iterating on the designs of models and domains for these systems challenging due to the limited insight their definitions provide about the nature of the spaces themselves. A lack of available tools and techniques to assist in understanding expressive possibility spaces reinforces this problem. Furthermore, this lack of support leads to a lack of easily intuitable information about the kinds of play experiences these possibility spaces produce. As a result, designing the possibility spaces themselves and relating them to resulting player experiences, human or AI, is frequently done through human playtesting and observation which can leave large areas of these spaces unexplored and underutilized.

Automated playtesting for games has largely been focused on evaluating generated levels (Liapis et al. 2015; Powley et al. 2016; Holmgård et al. 2018) and game balance (Zook, Harrison, and Riedl 2015; Pfau et al. 2020, 2022) and emulating human decision making (Devlin et al. 2016; Holmgård et al. 2016). The difficulty of gaining insights about AI systems, behaviors, and potential play experiences through purely human testing drove us towards developing methods of automated playtesting specifically for expressive models. With our goal in mind, we view core principles of automated playtesting as applicable to expressive models, helping us expose information about the expressive possibility spaces created by these models in more intuitive ways than currently exist. Interest in pursuing more player experienced approaches to automated playtesting is increasing, as recent work in the field illustrates (Barthet et al. 2022). Through automated playtesting we can gain a more complete picture of expressive possibility spaces than traditional playtesting approaches would reveal. In turn, enabling more deliberate design iteration and better understanding the implications changes to these spaces have on the player experience.

In this paper we propose Expressive Response Curves (ERC) as the base unit of our approach to automated playtesting. An ERC represents a sequence of character actions and states, in an expressive possibility space. Individual ERCs are then combined to create maps of the possibility spaces defined by AI models of expression. In doing so, *ERC maps* can surface information about how and when interaction and feedback is presented to a human player as they interact with the system or express themselves using the system. This mapping, therefore, allows us to analyze and ultimately characterize particular expressive possibility spaces with regards to the player experiences they afford. The focus of this paper is on the application of ERCs to the expressive possibility spaces created by a model of character expression, but ERCs applicable to expressive possibility spaces created from other sources like emergent narrative systems (Kybartas, Verbrugge, and Lessard 2020; Osborn, Samuel, and Mateas 2018).

Furthermore, as a sequence of potential actions and timings in relation to state, ERCs can surface information about moment-to-moment player actions and feedback. This allows us to expose how the player interacts system at the level

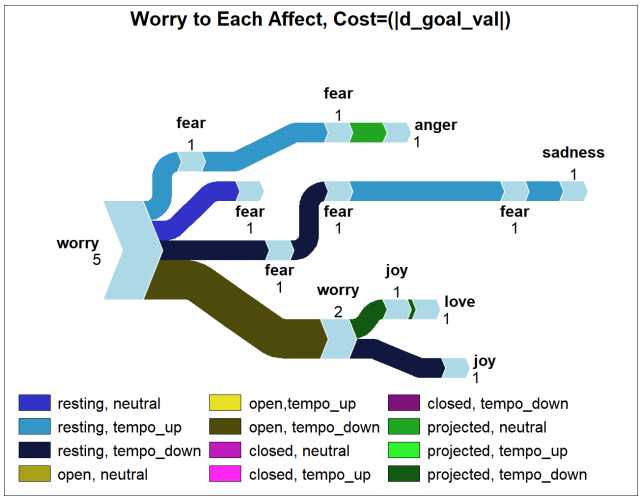

Figure 1: A collection of ERCs, creating an ERC map, where each flow path represents a single ERC and path from expressing worry to a different affect in the test domain of Puppitor. The same moves can be used as part of a sequence of actions to express different affects in Puppitor.

of character responsiveness to their actions. In this sense, therefore, ERCs allow us to visualize what is the equivalent of Game Feel (Swink 2008) for expressive character AI. This moves us closer to being able to directly study the connection between an expressive AI system's functionality and player experience and perception, and the dynamics between them.

We describe our approach to creating and using ERCs and present a case study of its application to Puppitor (Junius et al. 2022), an embedded domain specific language that defines the expressive qualities of a character to allow human and AI players to collaboratively perform scenes within narrative experiences. We chose Puppitor for this case study as it is a lightweight and flexible framework that suits our purpose of creating a complex expressive space in a human authorable format. Additionally Puppitor is a freely available, publicly accessible software library which has been used as part of a playable experience, from which we extracted rulesets to conduct our case study. Finally, we discuss the results of the case study and the implications ERCs have for the development of AI characters and performance.

ERC is not a recommendation tool or an approach for providing explicit suggestions to designers about the spaces they create. Rather, it is a method to allow them to make more informed decisions about the possibility spaces they are creating and how their designs could be perceived by a player.

Related Work

Our approach to creating ERCs is rooted in using concepts from game feel, namely response envelopes (Swink 2008) and juice (Gabler et al. 2005) as an alternative foundation to automated playtesting. The analytical aspect of ERCs draws inspiration from expressive range analysis (Smith and Whitehead 2010) as an existing approach to characterizing the possibility spaces created by generative systems. Finally, we chose to use A* (Hart, Nilsson, and Raphael 1968) as part of the approach to creating ERCs over other popular algorithms like Monte Carlo Tree Search (Chaslot et al. 2006) due to its more consistent and reproducible behavior. Additionally, A* has built in path generating functionality which is important for the creation of ERCs, as they are paths through expressive possibility space, and their analysis to characterize the expressive possibility space's affordances for player experience as inspired by game feel.

Automated playtesting has grown from the desire to have rapid evaluation of generated content such as video game levels (Liapis et al. 2015; Powley et al. 2016; Holmgård et al. 2018), game balance (Zook, Harrison, and Riedl 2015; Pfau et al. 2020, 2022), and skill progression (Horn et al. 2018). When used for the evaluation of generated levels, personas capturing human playstles are frequently used (Liapis et al. 2015; Ariyurek, Surer, and Betin-Can 2022). Monte Carlo Tree Search (MCTS), a best-first search algorithm developed for general gameplaying (Chaslot et al. 2006, 2008), is commonly used to drive automated playtesters (Zook, Harrison, and Riedl 2015; Keehl and Smith 2018; Horn et al. 2018; Holmgård et al. 2018; Mugrai et al. 2019) as it requires no domain knowledge to be integrated into the algorithm (Jacobsen, Greve, and Togelius 2014). Even with this benefit, MCTS does require some adaptation for the purpose of human-like playtesting. While the algorithm's high-level reasoning of trying to choose the best available move by predicting the outcomes does resemble parts of human decision-making processes (Holmgård et al. 2018), the commonly used selection formula, Upper Confidence Bound for Trees (UCT), frequently leads to inhuman choices for game moves (Devlin et al. 2016). Additionally, MCTS's nature as an aggregator of random actions makes its exact sequences of actions difficult to consistently reproduce, meaning for our purposes of creating ERCs, it is unsuited to our goal of finding reproducible paths through expressive possibility space.

Expressive range analysis (ERA) is a method of describing the possibility space of a generator, often a level generator, based on the artifacts it produces (Smith and Whitehead 2010). The existence of ERA creates the potential for analysis of other possibility spaces, including the expressive ones we describe in this paper or the results of automated playtesting (Agarwal et al. 2020). ERA has become one of the foundations for qualitative exploration and understanding the range of artifacts produced by a generator (Summerville 2018). The core of this approach to analysis is the selection of metrics to describe a level, each corresponding to an axis, to allow the for the visualization of the metric scores of a set of generated artifacts (Smith and Whitehead 2010). ERA has been incorporated into design support tools like Danesh (Cook et al. 2021) which will automatically analyze a set of generated artifacts as part of its goals for exposing the contours of generators.

With our goal of exposing the responsiveness of expressive systems to designers through automated playtesting, we use game feel (Swink 2008) as the foundation of our

method's understanding of the effects granular, low-level interactions have on player experience. Game feel describes the way interacting with a game is built out of numerous pieces coming together, from the ways an individual button press manifests on screen and through speakers to the ways giving a player goals alters their relationship to individual tasks (Swink 2008). The two most relevant aspects of game feel to the discussion of physical character expression are response metrics and low-level rules. When describing the relationship between input and feedback, Swink identifies the stages of attack, sustain, and release (ASR) as an approach to understanding the way a game interprets and responds to input (Swink 2008). Swink discusses ASR almost exclusively in relation to the ways button presses correspond to the responses of an on screen character. Low-level rules in turn define the physical relationships between entities, such as why larger enemies tend to take more damage to kill or bigger, heavier moves in fighting games tend to do more damage than lighter, faster ones (Swink 2008).

Additionally, juice is a subset of game feel focused on cascading audiovisual feedback in response to player actions and input (Gabler et al. 2005) and our usage of ERCs can provide insight about where juice may ideally be applied to heighten the play experience. With that said, the specifics of juicy design are highly dependent on the type of experience being developed or discussed (Hicks et al. 2018). Interestingly, with as much focus as juice has received as part of game feel and feedback mechanisms, it does not necessarily improve player performance in games and too much or ill considered incorporation of juicy elements can slightly detract from their use as feedback about performance (Hicks et al. 2019). While a discussion of juice as a component of interacting with AI controlled characters is beyond the scope of this work, ERC can help guide designers to places where incorporating such elements would be beneficial to the player experience.

We chose to use A* as the technical foundation for creating ERCs because it is a heuristic graph search and pathfinding algorithm. A* uses a breadth-first approach to visiting graph nodes and a priority queue based on cost and heuristic values to guide the search towards a specific goal (Hart, Nilsson, and Raphael 1968). Unlike breadth-first search, A* is not guaranteed to visit every node in a graph, nor is it guaranteed to find an optimal path without an admissible heuristic, a heuristic function which never over-estimates the distance to the goal (Hart, Nilsson, and Raphael 1968). The algorithm searches a given graph from a chosen starting point, towards a specified goal, using a heuristic to capture domain knowledge about the graph's domain to help prioritize promising nodes. From the starting node, each adjacent node is processed and added to a priority queue if it has not already been visited or the cost to reach the node is less than the previous visit. The node's priority is then determined by the cost to reach the node and the heuristic's estimated distance to the goal from the node. The highest priority node is then taken from the queue and the process is repeated until the goal is found. A* has been used in game related domains like path planning (Higgins 2002), action planning (Orkin 2005), and general game playing (Jacobsen, Greve, and Togelius 2014) thanks to its nature as a heuristic search allowing for flexibility across domains. In addition, the paths A* finds are more easily reproducible than other algorithms commonly used in game AI like MCTS. Both of these features allow ERCs to have a common technical basis which can be adapted to domains beyond those we consider explicitly in this paper.

Methodology

The method of creating and analyzing ERCs is focused on characterizing a possibility space through repeated searches to build a map of the locations in a sequence of actions and state changes that have noticeable and human perceptible feedback. Combining individual ERCs into a map of expressive possibility space exposes the complexity and responsiveness of actions required for a player to move between different expressive states described by the possibility space. Points in these possibility spaces are visited in sequence, with the path between them connecting them temporally to each other. Any location where a human perceptible change occurs in one of these sequences is then highlighted.

The steps for building and analyzing ERCs is as follows:

- **Translating the domain to A*:** Modify the chosen expressive possibility space and turn it into a graph domain to allow A* search to be used.
- **Adapting A* to the domain:** A* is a heuristic search and therefore has its cost and heuristic functions informed by the specifics of the domain to create ERCs.
- **Running searches:** Starting points for A* searches are chosen and search goals are defined in line with the domain knowledge representation to generate paths for the creation and analysis of ERCs.
- **Analyzing paths:** The paths found by A* are analyzed for their length, complexity, and responsiveness to player input and compared with each other to characterize the possibility space.

Translating the Domain to A*

To allow A* to operate over an expressive possibility space domain, its features must be translated into a graph format that the search algorithm can understand. This translation process focuses on identifying the elements of a given expressive possibility space that correspond to nodes and edges of a graph. A node defined by whichever domain elements describe a point in the possibility space. They contain a particular state representation, such as the relative values of each available expression and which expression is performed based on those values. An edge is made up of the elements of a possibility space which facilitate moving between the points it contains. These are represented as actions which have an effect on the state representation contained within a node, such as raising or decreasing the relative values of some of the expressions in a node. The exact details of this translation process are dependent on the way a given possibility space is defined. Which elements of a domain correspond to each graph element may not be obvious if concepts like state or state changes are not defined. We give a detailed example of the creation and application of ERCs in The case study sections.

Adapting A* to the Domain

Once a graph form of the possibility space is created, A* must be adapted to work with this new domain. This adaptation process focuses on defining the node representation, calculating edge costs, designing heuristic functions, and finally identifying abstractions necessary to the search, with the ultimate goal being the creation of an A* search which outputs useful and effective ERCs.

The representation of nodes able to produce ERCs requires that human perceivable changes to state be made primary components of a node. The translation of an expressive possibility space into a graph will not necessarily expose these state elements in an obvious way. For the purpose of creating ERCs, nodes must also highlight the components of the state which are feedback for the player. By highlighting these elements and making them primary components of the nodes, the sequences of actions found by A* search expose the perceived responsiveness of an expressive possibility space.

Edge costs are required to guide A* searches towards their goal and as a result can have noticeable effects on which sequences of actions A* chooses to perform, potentially making for more variance between individual ERCs. For this reason, costs are calculated during the search rather than being solely a fixed property of the graph definition. This decision allows the behavior of A* to be directed to prioritize different features of the domain and more effectively explore different permutations of action sequences. As a result the cost calculation must factor in the complexity and/or desirability of available actions. Different cost calculations then can be used to produce a wider range of individual ERCs to describe qualitatively distinct paths through parts of a domain's expressive possibility space, allowing for more comprehensive ERC maps to be created.

Creating a heuristic for A* in the domain expressive possibility space requires an understanding of the goal's representation, as a valid goal for the creation of an ERC can be as general as a component of feedback being exposed or as specific as an exact state representation. Because ERCs are focused on the human perceivable elements of expressive state, the goal of a search should reflect this priority. In turn, the heuristic function must have its calculations based on the state components responsible for tracking player feedback. As a result, when designing a heuristic function to produce ERCs, the distance estimate must include some understanding of player visible feedback as defined in the domain.

Finally, the complexity and scale of expressive possibility spaces can lead to searches becoming bogged down taking huge numbers of repeated actions to make minute changes to the state. To ease this calculation burden, any area of the search featuring long sequences of repeated action can be abstracted into single actions with cost and state changes to reflect the distance they cover, and to allow for much faster creation of individual ERCs.

Running Searches

Running searches for the purpose generating ERCs prioritizes finding starting points with distinct human perceivable differences in state. The goal of an individual search is another point in the expressive possibility space with perceivable differences to the starting point. Effective searches from a singular starting point cover all possible goals, whether or not a particular goal is in fact achievable from the chosen starting point.

Analyzing Paths

The process of creating a map of ERCs begins with choosing a starting point and an expressive goal to be found. The sequence of actions to reach this goal is then created using the search as described previously. Next, any point in the sequence of actions where a new action is taken or shows feedback visible to a player is highlighted. These steps are repeated for the same starting point for each possible expressive goal: creating a new ERC and highlighting the points of change in the found path. The individual ERCs using the same starting point are then combined to make a map of the expressive space between that starting point and all possible expressive goals in the possibility space.

The reason for highlighting changes in state, both player and system controlled, stems from Swink's response envelopes, which expose the relationship between a human player's input and the game responding with its own feedback (movement, animation, etc.) as a function of the time it takes for the game to complete a state transition. An ERC can then show the responsiveness of an area of a domain by exposing the length of time it takes for the state to respond to player input. This measurement of time can be as fine grained as the number of frames drawn to the screen, with the higher the frame count (or any other time measurement) between a player action and feedback from the state corresponding to lower responsiveness to the action taken.

Path length itself can be a measure of responsiveness and complexity. A lengthy path that contains long sequences of perceptually similar nodes, especially if their edges are distinct, is an indicator of an unresponsive area of the expressive possibility space, the expressive equivalent of the 1000 bowls of oatmeal problem (Compton 2016; Rabii and Cook 2023). Whereas a long path with many perceptually distinct nodes with multiple different kinds of edges connecting them is an indicator of complexity when trying to reach a particular part of expressive space from a given starting point.

Case Study: Puppitor Rulesets

Creating ERCs to analyze the expressive possibility space created by Puppitor character rulesets required us to translate Puppitor into a more easily searchable domain. Our chosen rules are extracted from *Tracks in Snow* (*TiS*) (June et al. 2021), a visual novel designed to showcase Puppitor (Junius et al. 2022). This process focused on identifying which elements of the domain are relevant to state representation and the actions which allow movement between expressive states, then converting them into a graph structure for use with A*. For this reason we begin this section with a summary of Puppitor as an expressive domain, then discuss our translation choices, approach to applying domain knowledge

to A*, and reasoning behind the creation of particular sets of paths through Puppitor's possibility space.

Puppitor

Puppitor is an embedded domain specific language that defines the expressive qualities of a character to allow human and AI players to collaboratively perform scenes within narrative experiences (Junius et al. 2022). The system creates a framework to map a character's physical actions to expressions of emotional affect. These physical actions are defined as *actions* and *modifiers*, with only one of each respective category being allowed to be performed at any time (the *prevailing* action and modifier). These actions and modifiers can be mapped to buttons on a keyboard to facilitate human input:

```
actions:
    open_flow : N
    closed_flow : M
    projected_energy : B
    resting : None
modifiers:
    tempo_up : C
    tempo_down : Z
    neutral: None
```

In the above example, the action and modifier, taken from one of the *TiS* rulesets, when no buttons are pressed are resting and neutral respectively. When only the B key is pressed, the action and modifier are projected energy and neutral respectively.

Puppitor uses these actions and modifiers to map actions to expressions of emotional affect. An affect vector contains the set of affects which a character can expressed mapped to their relative values. Only one affect can be expressed at a time, corresponding to the affect with the highest magnitude, the *prevailing affect*, with logic to handle the possibility of multiple affects having the same value (Junius et al. 2022).

```
joy : 0.35
anger : 0.1
sadness : 0.54
worry : 0.77
fear : 0.54
love : 0.28
```

In the above example, the prevailing affect is worry. Each affect listed in an affect vector must have a corresponding entry in a character's rule file which defines the values to be added to the affect vector during an update. When a Puppitor update is applied to an affect vector in a game loop, the prevailing action's value is multiplied by the prevailing modifier's value (Junius et al. 2022). For example, if we take the entry for joy pictured below and apply perform the projected energy action and neutral modifier with the above affect vector, joy will then have the value of `0.0005 * 1.0` added, resulting in a final value of 0.3505.

```
"joy" : {
    "actions" : {
        "resting" : -0.0003,
        "open_flow" : 0.00004,
        "closed_flow" : -0.0005,
        "projected_energy" : 0.0005
    },
    "modifiers" : {
        "tempo_up" : 1.14,
        "tempo_down" : 0.5,
        "neutral" : 1.0
    },
    "adjacent_affects" : {
        "love" : 90,
        "worry" : 10
    },
    "equilibrium_point" : 0.35
},
```

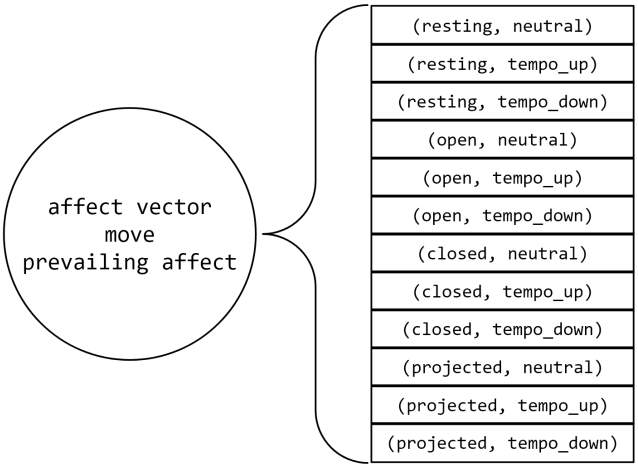

Figure 2: This diagram shows the contents of a single Puppitor graph node, the circle, and each outgoing edge, the rectangles.

Translating Puppitor into a Graph Domain

When looking at Puppitor's unmodified domain, we identify the affect vector and the actions and modifiers as the most important components of the system's definition of its expressive possibility space. We choose to translate affect vectors into graph nodes as each affect vector containing different values represents a unique point in Puppitor's expressive space. Actions and modifiers are what allow movement between affect vectors and more generally are what constrain the reachability of specific points in the possibility space. For our translation, we define an edge between to nodes of affect vectors as *moves*, the combination of Puppitor actions and modifiers.

With affect vectors being nodes, each node's outgoing edges correspond to every possible Puppitor move. There are four actions and three modifiers, making for a total of twelve moves and therefore twelve edges. Additionally, Puppitor allows any move to be performed even if it does not change the affect vector, so every node will have twelve outgoing edges. In turn, this means every node has at least twelve incoming edges, as nodes with minimized and maximized values in their affect vectors are reachable from huge numbers of

other nodes.

With this graph representation of Puppitor, with affect vectors as nodes and moves as edges, we can begin to apply domain knowledge to A*. The need for additional information to guide A* through Puppitor's rule space and ultimately facilitate the analysis of paths means the graph definition of affect vectors as nodes and moves as edges is a starting point for the process of creating ERCs of the system's rulesets.

Adapting A* Search to Puppitor

Adapting A* search for use with the graph derived from Puppitor focuses on four areas: the information in a node, the cost function, the heuristic, and the optimization needed to traverse a large, continuous space. The goal of each step of this process is to enable A* to output paths with the information required to produce ERCs.

An individual affect vector does not contain enough information for A* to effectively search between nodes and also does not easily expose the human perceivable parts of itself on its own. To enable A* search and the creation of ERCs, we must add the additional information of: the affect vector's prevailing affect and the move which led to the particular affect vector. Without the prevailing affect stored as part of the node, affect vectors which contain multiple values of the greatest magnitude become ambiguous about which affect a player sees when they reach the node. Additionally, multiple affect vectors may share identical values but none the less be perceptually distinct due to which moves were made to reach them. As a result, we also store the chosen move in the node. An example of the full node as represented to A* search would be:

```
affect vector: {
                joy : 0.3505,
                anger : 0.1,
                sadness : 0.54,
                worry : 0.77,
                fear : 0.54,
                love : 0.28
               },
action: projected_energy,
modifier: neutral,
prevailing affect: worry
```

Due to the complexity of the possibility space created by a Puppitor ruleset, having access to multiple edge cost functions can aid our goal of mapping as much of the space as possible. The first cost function we use is: $|\Delta goal\_val|$, which calculates edge cost as the magnitude of the change in the value of the goal affect as determined by the move performed. Using the example node, if our goal is to express joy, our cost of reaching this node would be 0.0005 as the move that got us to this specific node was `(projected_energy, neutral)` which has an update value of 0.0005 according to the ruleset we are using. The costs of edges, when summed together, enable us to calculate how far from the starting node we have traveled and compare the relative difficulty of reaching a particular node. In this case, as we have only moved one node from our starting point, the total cost for determining the priority of the example node is 0.0005.

The first cost function used generally leads to longer paths as changes of greater magnitude cost more, so the search will tend towards the smallest steps possible towards its goal. A second cost function we developed is: $||\Delta max\_val| - |\Delta goal\_val||$. This formula for cost incentivizes A* to focus on moves that provide the greatest change in both the currently prevailing affect and the goal value.

To create a heuristic function for Puppitor's domain, we must focus on the parts of the state preventing the goal affect from becoming the prevailing affect to produce an estimate of the distance to the search's goal. When pathfinding through expressive space, we do not prioritize reaching an exact affect vector, rather, our goal is to reach the nearest node where the prevailing affect is the goal affect. For this purpose we use the following logic:

max_nodes $= max($affect_vector$)$
max_value $= value($max_nodes$)$
if goal in max_nodes and curr_affect $\neq$ goal **then**
  heuristic_value $= big\_num$
else
  heuristic_value $= dist($max_value, goal_value$)$
end if
return heuristic_value

We consider the least desirable state to be one where the goal affect is among the affects with the highest value but is not the prevailing affect because any action taken that appears to increase the goal affect's value further will not in fact, which can cause the search to become stuck trying options which do not help it achieve its expressive goal. Therefore we assign it an extremely low priority as a result. If we are not in such an edge case, we estimate the distance to our goal as the difference between the highest valued affect and the goal affect. The heuristic value of our example node is calculated as the distance between worry (0.77) and joy (0.3505), resulting in a value of 0.4195. While a useful estimate, this heuristic is inadmissible as there is no guarantee of the value of the greatest magnitude remaining constant as Puppitor allows rules to update every available affect value and our rules used in this case study follow this pattern. However, we are not currently concerned with the optimality of a path, only if one exists. It is beyond the scope of this work to create an admimssible heuristic for such a new and under-explored domain.

Our final addition to A* for use with Puppitor's graph representation is the "navmeshification" of the domain. To equate this abstraction process we use with the use of A* to pathfind across geometric spaces abstracted as navmeshes. Puppitor rules can be written with very small update values. This authoring pattern is useful at creating continuous experiences over time but when searching across expressive space, it leads to huge numbers of repeated updates and slow traversal.

When A* searches over a unabstracted Puppitor ruleset, it is the equivalent of using A* for pathfinding over the high-detail environment meshes of an area rather than the

abstracted navmesh layer, i.e. moving between every tiny polygon of a cobblestone on a road. To abstract Puppitor updates, we consider each update to affect vector values a *step*. To avoid exploding the search space only to decide to perform the same action numerous steps in sequence, we can precalculate many steps in advance in the form of a multiplier.

With the example ruleset and affect vector, it would potentially take a minimum of 839 individual steps for the joy value to equal the worry value. Each step the search finds is fractionally small progress towards the goal and as a result many of the steps in the path will be performing the exact same move over and over, wasting time on calculations. If we instead pre-multiply a single step to instead be the equivalent of 90 steps, changing the update value to be 0.045 and one step from the initial value for joy to be 0.395, the minimum number of steps the search must calculate can be cut to approximately 10 steps.

Running Searches and Analyzing Paths through Expressive Space

For our search process, we picked six different starting affect vectors, each with a unique prevailing affect so we would cover multiple parts of the possibility space with even a low number of trials. We then told A* to search for each affect in our domain from each of these starting points. At most five unique paths will be found, as the search simply continues to perform the same move if it is already expressing the goal affect. In addition we ran two additional trials using an alternative cost function we developed during the process of this case study, the results of which can be found in figures 4 and 5.

The precalculation process requires us to reconstruct the full path and interpolate between the nodes A* discovers. Once A* finds a path between the starting point and the goal affect, nodes are added back to the node sequence to "unroll" the precalculations into the individual steps required to create a full path. The amount of reconstructed steps needed per node is based on the step multiplier originally used to precalculate the values. In our example, if the path A* creates is 10 steps, we must add an additional 89 steps between each of these nodes to account for the precalculation multiplier of 90 and interpolate between the initial value for joy of 0.35 and the next step's value of 0.395.

When reconstructing the path between steps, it is crucial to recalculate the prevailing affect with each step added. This part of the process is necessary as Puppitor rules allow a single update to change every value in an affect vector, meaning these reconstructed steps can have different prevailing affects than either node they are interpolating between. The changes in prevailing affect contained in these interstitial steps is hugely important to the creation of ERCs because they provide a far more detailed view of the way a Puppitor ruleset responds to a move than the abstracted A* nodes alone are able to show.

Once we have a complete path, we can highlight important points along the path, namely each step where the prevailing affect changes and each step where a new move is made. The highlighted path is now considered an ERC and can be used to characterize and analyze a Puppitor ruleset. The minute change in an affect vector's values are largely invisible until the prevailing affect changes. A new move being made changes the values being used to update the affect vector, and as a result, will alter the rates that each affect's relative strength change. Highlighting both of these steps allows us to analyze the responsiveness of a given ruleset, both for how quickly a desired affect can be expressed as well as the responsiveness of performing new moves. The shorter the number of steps between any of these changes, the more responsive the ruleset is to player input.

Case Study: ERC Map Analysis of a Puppitor Ruleset

The goal of our analysis of paths through expressive possibility space is simultaneously to characterize the human experience of specific expressive possibility spaces and to provide further insight into the ways an AI character understands and utilizes the space. The figures in this section all represent sets of individual possible paths through small corners of a single Puppitor ruleset. Each figure uses a single starting affect vector. Starting points that share a prevailing affect also share an affect vector. Each flow path shows the sequence of moves and affects to each other affect expressible from the starting point. These diagrams only represent individual paths to single possible ways of expressing affects and are not fully comprehensive captures of the entire expressive possibility space. Visualizing the full expressive possibility space of even a single Puppitor character is far too large and complicated an undertaking for the scope of this paper. Each node represents a change in prevailing affect. Each column of nodes represents a different point in time. Each flow path represents a unique move chosen. When multiple paths use the same move, the size of the corresponding flow is increased. The numerical values on each node correspond to the number of paths flowing into that unique node. Each terminal node represents the expressive goal of the A* search being reached.

For example in figure 1, there are three unique paths to expressing fear, each showing the expression of fear appear at a different point in time thanks to the different moves chosen. Additionally the two fear nodes sharing a time point in time are never the less unique as they were reached using different actions. This distinction is further shown by the differing moves and expressions flowing from them. Alternatively, the same move was chosen to express worry on two different paths. From this same point of worry, different moves lead to unique paths to joy and love. Due to the incentives provided by the cost function in this example, A* does not choose the moves which will allow it to express its goal in the shortest possible time frame. This decision-making is most obvious where to express fear when fear is the goal, A* chooses the `(resting, neutral)` move which is a slower path than the `(resting, tempo_up)` move.

The nature of the precalculations used to optimize A* mean that interstitial prevailing affects are not seen unless they persist into the next node found. This is clear in the case of figure 1 where joy briefly prevails before love, the latter

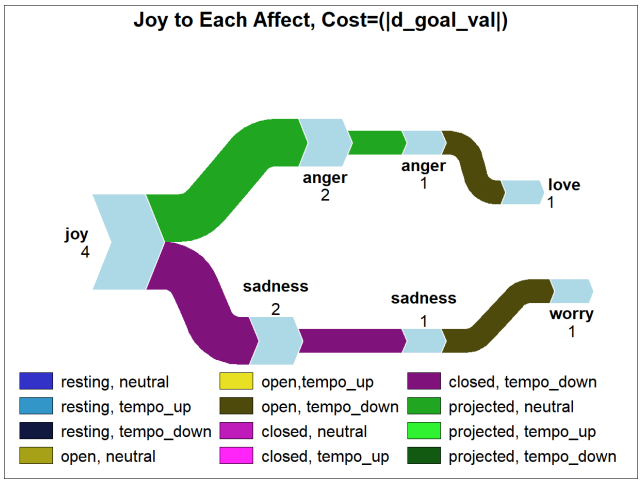

Figure 3: For some points in the expressive possibility space, affect values are such that moves can have very similar effects on multiple affects, hence why `(projected, tempo_up)` leads to both anger and love and `(closed, tempo_down)` leads to both sadness and worry. Also there is no path to expressing fear long enough for it to appear in a node visible to A*.

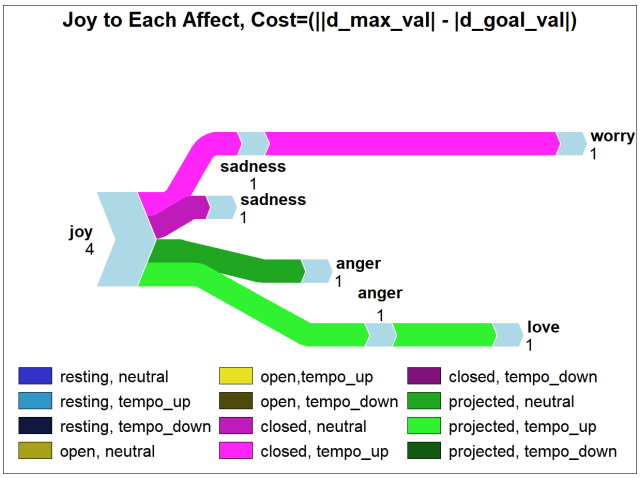

Figure 4: Altering the cost function leads to fully unique paths to each other expressible affect when starting at the same point of worry in the possibility space. These particular paths all exist inside of one precalculated step of A*, hence the lack of variation in moves and no shared points in time.

of which is the prevailing affect of the next node. Even if A* doesn't pay attention to these interstitial affects, they help give a better idea of the play experience around a particular point in the expressive possibility space.

Creating ERCs can also help find holes in the possibility space that may not be easy to intuit from rules. In the case of figure 3, the search did not find any path from this particular joy node to a node with the prevailing affect of fear. This particular ruleset was designed to make fear mostly only briefly prevail when moving between other affects, so its lack of presence is expected. In other cases, having ERCs available can help expose these gaps in the possibility space and allow for intentional decisions to be made about them earlier in the development process.

Using ERC to visualize the differences in searches, it becomes clear that the cost function of A* has a noticeable effect on the paths through the expressive possibility space. This change is most obvious in comparing figures 3, which uses the $|\Delta goal\_val|$ cost, and 4, which uses the $||\Delta max\_val| - |\Delta goal\_val||$ cost, as every affect able to be expressed has a unique set of moves to reach it in the latter, whereas there are only two paths in total in the former. The point of these comparisons is not to provide a recommendation for which formula produces better performances. Rather, ERC is meant to help inform the iteration and development of expressive characters by providing further insight into both the contours of expressive possibility space and the ways AI characters understand the domains they operate in.

Discussion and Future Work

By creating ERCs using Puppitor's character rulesets, we have been able to gain significantly more insight into the affordances of the system as an expressive domain for players and developers working in this area. On the player-oriented side, the use of ERCs can help identify overly homogeneous areas of a Puppitor ruleset and affects that are complex or impossible to express from certain points in a ruleset. For developers, having a fast way to view the effects that changes in domain knowledge and representation have on AI characters allows for much more deliberate iteration.

Our initial phase of analysis was focused on evaluating the responsiveness of ERCs created from a Puppitor ruleset. We define this property as the property of pathways which only require a single move to be made to see a change in the prevailing affect. The generated maps (figures 1, 3 , 4, and 5) show that the chosen ruleset often exhibited responsiveness in the individual ERCs, where individual move choices frequently led to changes in prevailing affect before a new move was considered. The starting points we chose for our figures, worry and joy, were chosen to show the effect the values inside an affect vector have on the difficulty and responsiveness of certain affects. We designed the worry starting point to make expression of a subset of affects difficult, shown in figures 1 and 5. Even then, the only affect goal which required A* to perform a single action for multiple nodes in its search was sadness. By evaluating a Puppitor ruleset by the responsiveness of the ERCs it produces, we can highlight the relative player experience of expressing affects from different locations in the possibility space.

The creation and analysis of ERCs for Puppitor enabled informed iteration of domain knowledge for A* as well as garnered insight into the way that domain knowledge changes the expressive performance of the search. We originally only used $|\Delta goal\_val|$ as the cost function to produce ERCs, but noticed that it rarely made paths containing obvious move choices when comparing its decisions with the ruleset definition. These comparisons led us to the re-

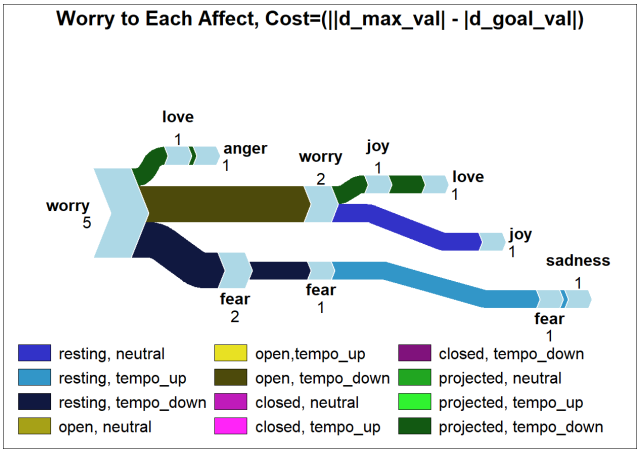

Figure 5: A different cost function doesn't always lead to dramatically different paths through the possibility space, but affects that can be easily expressed are much more likely to be found early in the search.

alization that to gain more full view of even these small sections of expressive possibility space, a single set of domain knowledge would not be sufficient, hence our usage of $||\Delta max\_val| - |\Delta goal\_val||$ as an additional cost function. With the two cost functions side by side, we could then characterize the performance styles they preferred, with the first ($|\Delta goal\_val|$) preferring more subtle moves and the second ($||\Delta max\_val| - |\Delta goal\_val||$) preferring slightly more active moves.

Comprehensively visualizing the information provided by ERCs is a challenge due to the nuances in the sequences of actions and their relationship to the surfaced player feedback present even in smaller domains like Puppitor. Knowing this, we note that while the Sankey diagrams our analysis of ERCs produced prove useful in illustrating the flow between affects and their relationship to player actions, these diagrams abstract away some important information for analyzing game feel. When describing these diagrams, we stated that nodes within the same column occur at the same point in time. The Sankey diagrams do maintain some of the information about the progression of time through nodes' column positions, but they flatten the details about the exact progression over time. This is especially notable for both worry trials (shown in figures 1 and 5). The diagrams cannot accurately capture the fact that the transition from worry to joy when performing the `(projected, tempo_down)` move occurs in a single step. As a result, further work must be done to enable the creation of more comprehensive visualizations of expressive possibility spaces.

With this in mind we want to highlight the complexity of the expressive possibility spaces we discuss in this paper or, more broadly, the difficulty of visualizing playtesting data (Agarwal et al. 2020). While we have been able to visualize small pieces of a Puppitor ruleset, current widely available visualization techniques limit us to a single starting point per diagram. The combinatorial explosion of even two or three starting points in a single diagram quickly renders them unusable for the purpose of analysis. Mapping every path through an expressive possibility space, and importantly presenting such paths in a user-friendly way, is an open research problem and far beyond the scope of this paper. With that said, ideally we will eventually be able to create tools to allow the full searching and mapping of expressive possibility spaces and effectively illustrate the relationships between clusters of paths. This fits the broader trend of games being incredibly difficult artifacts to adapt to data visualization. We plan to pursue the development of more effective visualization techniques for expressive possibility spaces in the future.

Puppitor provided an ideal testing platform for the creation and analysis of ERCs. It features clearly defined states with a well defined relationship to actions a player could perform, complete with values that could easily be turned into domain information for A* search. This clarity in the original domain made for a relatively straightforward translation process, but this will not necessarily hold true for other current and future expressive domains which ERCs may be useful to. Additionally, Puppitor has a built in way of signaling changes in its state that are human perceptible, its prevailing affects. Rather than having to derive a novel method for discovering important state transitions, Puppitor's prevailing affect system make this a core feature of its interface. The system's affordance therefore lightens the translation and adaptation burden by removing the need to synthesize a new approach to identifying human perceivable state transitions. Through our experience creating ERCs for Puppitor, we recognize that not all domains which can benefit from ERCs would have these features built in. As our current work with ERCs shows, we hope to encourage more systems to expose and highlight state changes that impact player experience.

Conclusion

In this paper we describe a method of automated playtesting which produces expressive response curves (ERCs) which can then be combined into a map to characterize the features of an expressive possibility space. Through our case study, we demonstrated that the insights provided by ERC maps allow for more informed reasoning about the domain on the part of designers, which in turn can lead to new ways of guiding A* performance within an expressive possibilty space. Combined with the more intuitive view of an expressive possibility space facilitated by ERCs, we can gain a far better understanding of the complexity of expressive domains and the nuances of their relationship to the development of AI characters focused on performance and expression. We hope that our work with ERCs inspires more work in the future focused on advancing our understanding of the player experience created by expressive AI models.

References

Agarwal, S.; Herrmann, C.; Wallner, G.; and Beck, F. 2020. Visualizing ai playtesting data of 2d side-scrolling games. In *2020 IEEE Conference on Games (CoG)*, 572–575. IEEE.

Ariyurek, S.; Surer, E.; and Betin-Can, A. 2022. Playtesting: What is Beyond Personas. *IEEE Transactions on Games*.

Barthet, M.; Khalifa, A.; Liapis, A.; and Yannakakis, G. 2022. Generative personas that behave and experience like humans. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*, 1–10.

Chaslot, G.; Bakkes, S.; Szita, I.; and Spronck, P. 2008. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, 216–217.

Chaslot, G.; Saito, J.-T.; Bouzy, B.; Uiterwijk, J.; and Van Den Herik, H. J. 2006. Monte-carlo strategies for computer go. In *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, 83–91.

Compton, K. 2016. So you want to build a generator... Blog post accessed on 5/26/2023.

Cook, M.; Gow, J.; Smith, G.; and Colton, S. 2021. Danesh: Interactive tools for understanding procedural content generators. *IEEE Transactions on Games*, 14(3): 329–338.

Devlin, S.; Anspoka, A.; Sephton, N.; Cowling, P.; and Rollason, J. 2016. Combining gameplay data with monte carlo tree search to emulate human play. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 12, 16–22.

Gabler, K.; Gray, K.; Shodhan, S.; and Kucic, M. 2005. How to prototype a game in under 7 days. *Gamasutra, October*, 26.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Hicks, K.; Dickinson, P.; Holopainen, J.; Gerling, K.; et al. 2018. Good game feel: an empirically grounded framework for juicy design. In *Proceedings of the 2018 DiGRA International Conference: The Game is the Message. DiGRA*.

Hicks, K.; Gerling, K.; Dickinson, P.; and Vanden Abeele, V. 2019. Juicy game design: Understanding the impact of visual embellishments on player experience. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, 185–197.

Higgins, D. 2002. Generic A* pathfinding. *AI Game Programming Wisdom*, 114–121.

Holmgård, C.; Green, M. C.; Liapis, A.; and Togelius, J. 2018. Automated playtesting with procedural personas through MCTS with evolved heuristics. *IEEE Transactions on Games*, 11(4): 352–362.

Holmgård, C.; Liapis, A.; Togelius, J.; and Yannakakis, G. N. 2016. Evolving models of player decision making: Personas versus clones. *Entertainment Computing*, 16: 95–104.

Horn, B.; Miller, J.; Smith, G.; and Cooper, S. 2018. A Monte Carlo approach to skill-based automated playtesting. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 14, 166–172.

Jacobsen, E. J.; Greve, R.; and Togelius, J. 2014. Monte mario: platforming with mcts. In *Proceedings of the 2014 annual conference on genetic and evolutionary computation*, 293–300.

June, N.; Murray, R.; Marshall, C.; Duplantis, T.; Karth, I.; and Kreminski, M. 2021. Tracks in Snow. [PC Digital Download itch.io], USA: Quakefultales.

Junius, N.; Mateas, M.; Wardrip-Fruin, N.; and Carstensdottir, E. 2022. Playing with the Strings: Designing Puppitor as an Acting Interface for Digital Games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 18, 250–257.

Keehl, O.; and Smith, A. M. 2018. Monster Carlo: an MCTS-based framework for machine playtesting unity games. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8. IEEE.

Kybartas, Q.; Verbrugge, C.; and Lessard, J. 2020. Tension space analysis for emergent narrative. *IEEE Transactions on Games*, 13(2): 146–159.

Liapis, A.; Holmgård, C.; Yannakakis, G. N.; and Togelius, J. 2015. Procedural personas as critics for dungeon generation. In *Applications of Evolutionary Computation: 18th European Conference, EvoApplications 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings 18*, 331–343. Springer.

Mugrai, L.; Silva, F.; Holmgård, C.; and Togelius, J. 2019. Automated playtesting of matching tile games. In *2019 IEEE Conference on Games (CoG)*, 1–7. IEEE.

Orkin, J. 2005. Agent architecture considerations for real-time planning in games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 1, 105–110.

Osborn, J. C.; Samuel, B.; and Mateas, M. 2018. Visualizing the strategic landscape of arbitrary games. *Information Visualization*, 17(3): 196–217.

Pfau, J.; Liapis, A.; Volkmar, G.; Yannakakis, G. N.; and Malaka, R. 2020. Dungeons & replicants: automated game balancing via deep player behavior modeling. In *2020 IEEE Conference on Games (CoG)*, 431–438. IEEE.

Pfau, J.; Liapis, A.; Yannakakis, G. N.; and Malaka, R. 2022. Dungeons & replicants II: automated game balancing across multiple difficulty dimensions via deep player behavior modeling. *IEEE Transactions on Games*.

Powley, E. J.; Colton, S.; Gaudl, S.; Saunders, R.; and Nelson, M. J. 2016. Semi-automated level design via auto-playtesting for handheld casual game creation. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8. IEEE.

Rabii, Y.; and Cook, M. 2023. Why Oatmeal is Cheap: Kolmogorov Complexity and Procedural Generation. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*, 1–7.

Smith, G.; and Whitehead, J. 2010. Analyzing the expressive range of a level generator. In *Proceedings of the 2010 workshop on procedural content generation in games*, 1–7.

Summerville, A. 2018. Expanding expressive range: Evaluation methodologies for procedural content generation. In

Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, volume 14, 116–122.

Swink, S. 2008. *Game feel: a game designer's guide to virtual sensation*. CRC Press.

Zook, A.; Harrison, B.; and Riedl, M. O. 2015. Monte-carlo tree search for simulation-based strategy analysis. In *Proceedings of the 10th International Conference on the Foundations of Digital Games*.